University of Plymouth

PEARL **https://pearl.plymouth.ac.uk**

04 University of Plymouth Research Theses 01 Research Theses Main Collection

2020

PROACTIVE BIOMETRIC-ENABLED FORENSIC IMPRINTING SYSTEM

Alruban, A

http://hdl.handle.net/10026.1/15562

http://dx.doi.org/10.24382/986
University of Plymouth

All content in PEARL is protected by copyright law. Author manuscripts are made available in accordance with publisher policies. Please cite only the published version using the details provided on the item record or document. In the absence of an open licence (e.g. Creative Commons), permissions for further reuse of content should be sought from the publisher or author.

UNIVERSITY OF PLYMOUTH

PROACTIVE BIOMETRIC-ENABLED FORENSIC IMPRINTING SYSTEM

by

ABDULRAHMAN ALRUBAN

A thesis submitted to the University of Plymouth

in partial fulfilment for the degree of

DOCTOR OF PHILOSOPHY

School of Engineering, Computing and Mathematics

April 2020

Copyright Statement

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without the author's prior consent.

Copyright © 2020 Abdulrahman Alruban

Acknowledgements

First and foremost, all praise and gratitude are due to Allah Almighty, the All-Merciful, for helping me and facilitating me in tackling all the challenges throughout this PhD which could have constrained my study.

I am deeply indebted and most of my sincere thanks and appreciation go to my beloved parents, for their considerable help and support, kindness, abundant love, and prayers for my study and I ask Allah to reward them with the best. I would also like to take this opportunity to express my sincere gratitude to my brothers and sisters for their immeasurable love and encouragement through this important stage of my life. For my parents, brothers and sisters, I am forever grateful.

I also owe many thanks to my beautiful love my wife Mona and my children, Alwaleed, Albaraa, Mohammed and Abdullah, for their patience, endless support, and incredible care in assisting me throughout this endeavour. They all stood alongside me and provided me with an abundance of love and support, even when spending days, nights, and holidays without me. I really appreciate your endless support and help through this PhD journey. I am eternally grateful.

Of course, I would like to extend my most sincere thanks and my heartfelt appreciation to my supervision team, Professors Nathan Clarke, Fudong Li and Steven Furnell, for their guidance, support, wisdom, help and a sympathetic ear. Their experience and professionalism in various aspects, such as their critical thinking, publications and presentations, have been invaluable throughout my PhD journey and without their valuable comments and advice, I would not be able to make this a success, so thank you.

I would also like to express my thanks to my research colleagues at the Centre for Security, Communication and Network Research, Saud Alotaibi, and Abdulwahid Al Abdulwahid, who has been my motivation and inspiration and with whom I have held interesting discussions during this PhD journey.

I would like to take this opportunity to thank all my colleagues at Majmaah University in the Kingdom of Saudi Arabia for allowing me to take this great opportunity to complete my PhD degree and for their support and assistance.

Author's Declaration

At no time during the registration for the degree of Doctor of Philosophy has the author been registered for any other University award without prior agreement of the Doctoral College Quality Sub-Committee.

Work submitted for this research degree at the University of Plymouth has not formed part of any other degree either at the University of Plymouth or at another establishment.

This study was financed with the aid of a studentship from the Royal Embassy of the Kingdom of Saudi Arabia.

Relevant scientific seminars and conferences were regularly attended at which work was often presented and published.

Word count of thesis: 47,956

List of Publications:

Alruban, A., Clarke, N., Li, F. and Furnell, S., 2016, June. Proactive biometric-enabled forensic imprinting. In 2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security) (pp. 1-15). IEEE.

DOI: https://doi.org/10.1109/CyberSecPODS.2016.7502342

Alruban, A., Clarke, N., Li, F. and Furnell, S., 2016, 'Leveraging biometrics for insider misuse identification', International Journal on Cyber Situational Awareness (IJCSA), 1(7).

DOI: https://doi.org/10.22619/IJCSA

Clarke, N., Li, F., Alruban, A. and Furnell, S., 2017, January. Insider misuse identification using transparent biometrics. In Proceedings of the 50th Hawaii International Conference on System Sciences.

DOI: https://doi.org/10.24251/HICSS.2017.487

Alruban, A., Clarke, N., Li, F. and Furnell, S., 2017, August. Insider misuse attribution using biometrics. In Proceedings of the 12th International Conference on Availability, Reliability and Security (p. 42). ACM.

DOI: https://doi.org/10.1145/3098954.3103160

Alruban, A., Clarke, N., Li, F. and Furnell, S., 2018, September. Biometrically linking document leakage to the individuals responsible. In International Conference on Trust and Privacy in Digital Business (pp. 135-149). Springer, Cham.

DOI: https://doi.org/10.1007/978-3-319-98385-1_10

Alruban, A. and Clarke, N., 2016. Method of Associating A Person with A Digital Object. GB Patent filing GB1609673, 7.

Signed: Abdulrahman

Date : 8 April 2020

Abstract

PROACTIVE BIOMETRIC-ENABLED FORENSIC IMPRINTING SYSTEM

ABDULRAHMAN ALRUBAN (MSc)

Insider threats are a significant security issue. The last decade has witnessed countless instances of data loss and exposure in which leaked data have become publicly available and easily accessible. Losing or disclosing sensitive data or confidential information may cause substantial financial and reputational damage to a company. Therefore, preventing or responding to such incidents has become a challenging task. Whilst more recent research has focused explicitly on the problem of insider misuse, it has tended to concentrate on the information itself—either through its protection or approaches to detecting leakage. Although digital forensics has become a de facto standard in the investigation of criminal activities, a fundamental problem is not being able to associate a specific person with particular electronic evidence, especially when stolen credentials and the Trojan defence are two commonly cited arguments. Thus, it is apparent that there is an urgent requirement to develop a more innovative and robust technique that can more inextricably link the use of information (e.g., images and documents) to the users who access and use them. Therefore, this research project investigates the role that transparent and multimodal biometrics could play in providing this link by leveraging individuals' biometric information for the attribution of insider misuse identification.

This thesis examines the existing literature in the domain of data loss prevention, detection, and proactive digital forensics, which includes traceability techniques. The aim is to develop the current state of the art, having identified a gap in the literature, which this research has attempted to investigate and provide a possible solution. Although most of the existing methods and tools used by investigators to conduct examinations of digital crime help significantly in collecting, analysing and presenting digital evidence, essential to this process is that investigators establish a link between the notable/stolen digital object and the identity of the individual who used it; as opposed to merely using an electronic record or a log that indicates that the user interacted with the object in question (evidence). Therefore, the proposed approach in this study seeks to provide a novel

technique that enables capturing individual's biometric identifiers/signals (e.g. face or keystroke dynamics) and embedding them into the digital objects users are interacting with. This is achieved by developing two modes—a centralised or decentralised manner. The centralised approach stores the mapped information alongside digital object identifiers in a centralised storage repository; the decentralised approach seeks to overcome the need for centralised storage by embedding all the necessary information within the digital object itself. Moreover, no explicit biometric information is stored, as only the correlation that points to those locations within the imprinted object is preserved.

Comprehensive experiments conducted to assess the proposed approach show that it is highly possible to establish this correlation even when the original version of the examined object has undergone significant modification. In many scenarios, such as changing or removing part of an image or document, including words and sentences, it was possible to extract and reconstruct the correlated biometric information from a modified object with a high success rate. A reconstruction of the feature vector from unmodified images was possible using the generated imprints with 100% accuracy. This was achieved easily by reversing the imprinting processes. Under a modification attack, in which the imprinted object is manipulated, at least one imprinted feature vector was successfully retrieved from an average of 97 out of 100 images, even when the modification percentage was as high as 80%. For the decentralised approach, the initial experimental results showed that it was possible to retrieve the embedded biometric signals successfully, even when the file (i.e., image) had had 75% of its original status modified. The research has proposed and validated a number of approaches to the embedding of biometric data within digital objects to enable successful user attribution of information leakage attacks.

Table of Contents

List of Figures

List of Tables

List of Equations

Glossary

ASCII	-	American Standard Code for Information Interchange
AES	-	Advanced Encryption Standard
BIOFI	-	Biometric-enabled forensic imprinting system
CCTV	-	Closed-circuit television
DCT	-	Discrete cosine transform
DLD/P	-	Data loss detection and prevention
DLP	-	Data loss prevention
DOCX	-	Microsoft Word documents
EER	-	Equal error rate
FAR	-	False acceptance rate
FNIR	-	False negative identification rate
FPIR	-	False positive identification rate
FRR	-	False rejection rate
FV	-	Feature vector
HR	-	Heart rate
HRV	-	Heart rate variability
IP	-	Internet Protocol
IRILD	-	Information-retrieval-based information leakage detection
LDA	-	Linear discriminant analysis
LSA	-	Latent semantic analysis
LSB	-	Least significant bit
LSH	-	Locality sensitive hashing
MSE	-	Mean squared error
OCR	-	Optical character recognition
PCA	-	Principal component analysis

PDF	-	Portable document format
PSNR	-	Peak signal-to-noise ratio
RDBMS	-	Relational database management system
SHA	-	Secure Hash Algorithm
SQL	-	Structured Query Language
SVD	-	Singular value decomposition
SVM	-	Support vector machine
TAR	-	True acceptance rate
TLSH	-	Trend micro locality sensitive hash
TPIR	-	True positive identification rate
TRR	-	True rejection rate

1 Introduction

1.1 Introduction

Insider threats to enterprises have become widespread in the last decade (Liu *et al.*, 2018; Safa *et al.*, 2018) and have, therefore, been considered as a crucial security issue by many recent research studies (Shabtai, Elovici and Rokach, 2012a; Collins *et al.*, 2013; Huth *et al.*, 2013; L. Liu *et al.*, 2018; Nourian and Madnick, 2018). In addition, insiders who have legitimate access to an organisation's internal systems and databases have the advantage of accessing all kinds of information—including those classified as confidential—as they possess the appropriate permissions and rights. A study found that more than 300,000 internal security breaches took place in UK businesses in 2013 (IS Decisions, 2014). More recently, according to the 2018 Data Breach Investigations Report by Verizon, which included 2,216 data breaches, one-quarter of the attacks that occurred in organisations were mainly driven by misuse, mistakes, espionage or financial gain (Verizon, 2018). Such breaches lead to substantial damage to the exploited body as a result of the loss or disclosure of sensitive and confidential intellectual property. In particular, when the exposure originates from an authorised person (employee, contractor, etc.) who misuses the advantage of privileged and legitimate access to the firm's internal resources, this facilitates potential access to restricted areas, in comparison with the approach of outsiders who do not have the prior knowledge possessed by insiders. This is because insiders are more likely to be able to bypass security controls compared with

outsiders who supposedly have limited knowledge about the internal infrastructure. As a result, insiders pose a significantly greater threat to organisations than outsiders (Stanton *et al.*, 2005; Colwill, 2009).

According to Felix Gaehtgens, a research director at Gartner, in 2015, fewer than 5% of organisations were actually tracking and reviewing privileged activities, while the remainder were, at best, controlling access and logging when, where and by whom privileged access took place—but not what was actually done. Unlike those who monitor and evaluate the privileged activity, those who do not are at risk of being blindsided by insider threats, malicious users or errors that create significant threats (Gartner Inc., 2016; L. Liu *et al.*, 2018).

Digital forensics aims to produce and test a hypothesis about who did what, where, when and how in relation to the incident under investigation. Indeed, existing methods and tools used by investigators to conduct examinations of digital crime are of significant help in collecting, analysing and presenting digital evidence (Carbone, 2014; Widup, 2014; SANS Institute, 2016a). However, this remains a challenging task, because it is currently difficult for digital forensic professionals and investigators to prove beyond a reasonable doubt in a court of law that a specific human being has used the specific identity of a digital subject at a particular time (Shavers, 2013; Brown, 2015; Vincze, 2016). In many cases, the furthest point that a forensic investigation can reach by analysing electronic evidence—with the aim of tracing the origin of the crime committed—is the machine or Internet Protocol (IP) address from which the crime was committed. In such scenarios, criminals could argue and deny a charge by claiming that someone

else had used their computer, especially knowing that a lot of desk-based employees think that there is no security risk to their employer in sharing work login credentials (IS Decisions, 2018).

Having an approach that links information with those using it—beyond simple accounts—is required to provide the ability to connect an individual with the data being used. Most existing methods, such as digital watermarking or logs-based systems, establish correlation solely by a digital record of the object being monitored—not with the individual directly (Charbonneau and Simon, 2014; Nelson and Xie, 2014). As a possible solution to this problem, research has focused upon the use of biometrics that could provide such a link. Moreover, transparently capturing the user's biometrics and instantly generating a biometric imprint that correlates the user interaction with the object used could give rise to critical information that would help digital forensic investigators in answering the "who" question. Upon detection of misuse, the biometric signal can be recovered and this will identify the individual who sent or interacted with a given digital object—negating the need for expensive and time-consuming investigation of a system that could eventually lead to a digital record (e.g., username, IP address, computer ID, etc.).

1.2 Research Aim and Objectives

The main aim of this research is to develop *an efficient and robust proactive biometric-based forensic system that can inextricably link the use of information (evidence) to the individual users who access and use it.* In order to achieve this aim, the following research objectives were established:

1. To establish a current state-of-the-art understanding of data loss prevention, detection and proactive digital forensic techniques focusing on insider misuse, with a view to identifying and assessing the necessary attributes to enable a proactive and innovative digital forensic approach.
2. To propose a novel biometric-based technique that works proactively to establish a correlation between a digital object and the individual who interacts with it. The following investigations were carried out:
    2.1. Investigation of differing embedding processes, including leveraging steganographic techniques such as null and grille cipher methods.
    2.2. Investigation of how the correlation between the biometric signal acquired and the object being monitored could be established with different file types, including images and text-based files.
    2.3. Investigation of the impact on biometric retrieval given differing levels of file modification (i.e., having a partial image, the removal of paragraphs, sentences, and words, and file format conversion).
    2.4. Investigation of the volume of biometric data that can be embedded and the impact upon retrieval (single and multimodal).
3. To identify the strengths and limitations of the proposed approach based upon the series of conducted investigations and experiments.

1.3 Thesis Structure

This thesis is organised into nine chapters. Chapter 1 introduces the research problem and outlines the overall research aim and its objectives, along with the structure of this report. The chapter concludes with a list of the works published during this research project.

Chapter 2 explores the definition of 'an insider' and the scale and cost of insider misuse. This chapter also provides background information about digital forensics in general, including the investigation process and the models that most forensic professionals follow when conducting a digital crime investigation. An overview of the current challenges in digital forensics is presented and the role of biometrics within digital forensics discussed.

Chapter 3 provides a literature review of related studies in the domain of data loss detection, prevention and proactive digital forensic techniques. The chapter concludes with a discussion section that identifies a gap that exists in the literature.

Chapter 4 identifies system actors, use cases and a relevant scenario to define, investigate, and evaluate how the system would react in normal operation. The chapter form the bases for the design requirement of the proposed system.

Chapter 5 introduces a proactive digital forensic biometric-based approach to the attribution of misuse via information leakage using biometrics and steganography. Two main methods (i.e., null cipher and grille cipher imprinting techniques) are

discussed in this chapter. The chapter ends with the main research questions to be investigated experimentally in the subsequent chapters.

Chapter 6 is the first of three chapters that consider an experimental investigation and presents a biometric-based null cipher technique using images by embedding individuals' biometric signals into image files, with a particular focus upon the ability to recover the biometric information under varying degrees of modification attack. The chapter also provides a detailed analysis in evaluating the performance of the approach used.

Chapter 7 presents an investigation conducted into a biometric-based grille cipher technique using images by linking a subject (i.e., computer user) with an object of interest (e.g., images) using the individual's biometric sample, such as a facial biometric, without modifying the object being imprinted. This investigation also develops a set of experiments that employ a grille cipher technique to generate the correlation that could identify the individual.

Chapter 8 presents an investigation into a biometric-based grille cipher technique using documents to attribute document misuse via information leakage using biometrics and a locality sensitive hashing scheme. A comprehensive set of experiments for the proposed approach are conducted, such as changing the file format and removing parts of the document, including words and sentences, to measure the effectiveness of the developed technique.

Chapter 9 provides conclusions and highlights the main contributions and achievements of the research project. The chapter also describes the limitations

identified during the project. The chapter ends with suggestions and scope for possible future work.

At the end of this thesis, scripts and codes for the developed algorithms are attached.

2 Insider Misuse and Digital Forensics

This chapter provides information about insiders and the problem of insider misuse, together with a brief background to the digital forensic investigation processes and models that most forensic professionals follow when conducting digital crime investigations. An overview of the current challenges in digital forensics is also presented, together with an analysis of the role that biometrics plays within the domain.

2.1 Introduction

It is deeply concerning for organisations when data exposure originates from an authorised individual (e.g., an employee or contractor) who misuses their legitimate access and the potential for adverse impacts is, in this case, typically higher than for access by outsiders (Moshinsky, 2017; Titcomb, 2017; WikiLeaks, 2017). Insiders are more likely to bypass security controls than outsiders, as the latter typically have limited knowledge of the internal infrastructure in a given case. Therefore, insiders pose a significant threat and identifying criminals, especially if the digital forensic process leads to the presentation of findings in legal proceedings, is a challenging and crucial task.

2.2 Insider Misuse

There is no unified definition of an ‘insider’ since it depends heavily upon the nature of the organisation and how its security perimeters are defined. It becomes even more complicated when an organisation has a range of employees who are involved in an incident. For instance, a company might have contractors,

subcontractors, part-time employees, business partners, and even outsourcers. The US Department of Homeland Security defines an 'insider' as "an individual with privileged access to an information technology system" (Hunker and Probst, 2011). This definition is acceptable when the insider threat occurs in an information technology (IT) context. However, it seems to ignore the question of whether the individual is trusted or not (Bishop and Gates, 2008). It also ignores other aspects of the issue, in addition to the IT resources. For instance, an employee might work in a testing laboratory, where it is possible to leak intellectual property (e.g., a documentary design or a prototype of a new secret product). This type of malicious activity can be carried out without the need for privileged access to IT resources. Therefore, limiting the definition simply to having privileged access to IT resources is excessively specific and is not necessarily appropriate to many other relevant scenarios.

Another definition defines an insider as any person who works within the security boundaries of an organisation (Patzakis, 2003). However, the revolution in mobile devices has enabled individuals to be freed from such security perimeters, and their tasks may be accomplished outside the organisation's physical site. Therefore, legitimate users might work inside or outside their institution's site via modern technology. Hence, it is difficult to have a generic definition of an insider, although the definition should be consistent with the context and environment.

2.2.1 Insider Threats

According to Cappelli et al. (2012) and Elmrabit, Yang and Yang (2015), insider threats can be categorised into three main types: insider Internet technology

sabotage, insider theft of intellectual property, and insider fraud. Most insider threat cases that occurred in the past can fit into one of these three areas, depending upon the intent of the malicious activity. Insider IT sabotage mostly includes those incidents in which the crime is committed by a technical user, for instance, a database, network or system administrator. In most documented cases, it was found that insider IT sabotage crimes, such as causing denial of service, were conducted as revenge against the victim organisation (Clark, 2016; Jackson, Choi and Gelfand, 2019).

The underlying motivation for insider theft of intellectual property is typically to steal something that belongs to the criminal's institution. Such a crime is typically committed by someone who has access to sensitive information and seeks to gain personal benefit from stealing intellectual property. For instance, an employee moving from one job to another in a different organisation might steal a software source code or novel product design in order to use it in a new job.

The third category of insider threat is insider fraud, which is usually carried out by low-level workers, such as a data entry worker, call centre representative or secretary. In this case, the individual is trying to gain some benefit from the crime committed for themselves or another organisation. An example of this type of insider threat is someone leaking confidential financial information, such as customer credit card details, to somebody who is not authorised to access those data. Typically, the common motivation behind this type of crime is financial gain. For the purpose of this research, the main focus is on insider theft of intellectual

property, specifically data leakage; the other categories are thus beyond the scope of this project.

2.2.2 Monitoring Users' Behaviour and the Law

When it comes to observing employees' activities, the law varies from one country to another. For example, in the UK, it is obligatory to make staff aware that they are being monitored (Gov.uk, 2015). Furthermore, employers must explain clearly the type of observation that is undertaken and its purpose. However, according to human rights law, there are two cases in which employers can monitor their staff without their knowledge: when someone is suspected of breaking the law and when making them aware of the observation would make it more difficult to detect a crime.

In the US, the laws for monitoring employees differ from state to state. For instance, some states, such as Delaware and Connecticut, regulate computer-use observation and require the employer to notify employees in advance when there is an active monitoring system in place. In other states, such as Michigan and Illinois, employers are prohibited from monitoring their staff concerning information related to "an employee's associations, political activities, publications, or communications of non-employment activities" (Huth, 2013). In general, most US states require employers to inform their employees of monitoring activity in advance.

2.3 Digital Forensic Investigation Processes

The science of digital forensics has existed for a long time, aiding organisations in investigating cybercrime. In the early days of digital forensics, professionals and practitioners tended to use the term *computer forensics* to describe the process of investigating a computer crime incident. Not until the emergence of other electronic and embedded devices, such as mobiles and smartphones, did forensic specialists move to call it the *digital forensics* domain. A modern and trending term that describes wider branches in the domain of digital forensics is *cyber forensics*, which encompasses digital and computer forensics. In general, all three terms are likely to be used as synonyms to describe the process and science of extracting information and data from electronic devices to serve as electronic evidence for proving and legally prosecuting digital crime. This process includes, but is not limited to, extracting relevant information from computers, smartphones, network devices, databases, and storage media.

One of the main aims of the digital forensic process is to produce and test a hypothesis about who did what, when, where and how in relation to the incident under investigation in a way that is legally acceptable to the courts. Indeed, the existing methods and tools used by investigators to conduct examinations of a digital crime help significantly in collecting, analysing and presenting digital evidence (Carbone, 2014; Widup, 2014; SANS Institute, 2016b). However, the question of who committed the crime is crucial, especially if the digital forensic process leads to the presentation of findings in a court of law (Valjarevic and Venter, 2012). Therefore, employing the digital forensic process within an

organisation in order to investigate crime incidents successfully is neither a straightforward nor trivial task. Such a process involves several practical and technical aspects, which, if not conducted properly, will significantly affect the forensic value of the investigation and the resulting evidence (Clarke, 2010). Most legal forensic practices are often post-incident investigations designed to gather and analyse information that assists in building a hypothesis to form a case of the law. Such action is typically called a reactive model, whereby the whole investigation takes place following the incident. What could help in identifying the source of a crime would be to prepare the environment by being proactive in putting appropriate processes and technologies in place in order to be proactive in aiding the post-incident investigation. This would significantly and effectively boost the success of the overall investigation process in proving guilt in a crime.

2.3.1 Reactive Model

A reactive investigation acts like a typical digital forensic investigation process. It mainly involves seven successive phases that forensic practitioners follow, as Figure 2.1 illustrates. The process starts with the identification phase and ends with a decision. The main phases are written along the top of the table. Each column contains items and steps for the corresponding category to which it belongs. However, according to the discussion during the first meeting of the Digital Forensic Research Workshop (DFRWS) in 2001, it was debated that not all the classes/phases were to be considered ‘forensic’. Those items in the centre of the table (highlighted in grey) are those open to the least confusion, although

there is still some argument about the use of the terms collection and preservation, and whether one is really a subclass of the other (Palmer, 2001).

Identification	Preservation	Collection	Examination	Analysis	Presentation	Decision
Event/Crime Detection	Case Management	Preservation	Preservation	Preservation	Documentation	
Resolve Signture	Imaging Technologies	Approved Methods	Traceability	Traceability	Expert Testimony	
Profile Detection	Chain of Custody	Approved Software	Validation Techniques	Statistical	Clarification	
Anomalous Detection	Time Synch.	Approved Hardware	Filtering Techniques	Protocols	Mission Impact Statement	
Complaints		Legal Authority	Pattern Matching	Data Mining	Recommended Countermeasure	
System Monitoring		Lossless Compression	Hidden Data Discovery	Timeline	Statistical Interpretation	
Audit Analysis		Sampling	Hidden Data Extraction	Link		
Etc.		Data Reduction		Spacial		
		Recovery Techniques				

Figure 2.1: Generic reactive digital forensic process model (Palmer, 2001)

Each of the phases is subdivided into various components to give a broad outline of the goal that particular phase is trying to achieve.

Identification includes the detection of an event or incident using existing tools, for example, intrusion detection techniques or monitoring systems. However, the identification process is not typically a digital forensics role but is required to begin the investigation process.

Preservation involves gathering relevant data. It is useful for the purposes of digital investigations by ensuring that none of the data is lost or tampered with but preserved according to the related procedures that are normally followed in an accredited digital forensics house.

Collection consists of reducing the preserved information to relevant data by appropriate hardware and software tools, using the right sampling techniques so that the target data are much more relevant than the preserved data in terms of the digital forensics examination purposes.

Examination includes close scrutiny of the data to find traces of a crime by searching for aspects such as hidden data, pattern matches and validating the data. During this phase, the digital evidence is evaluated to disclose data and reduce volume.

Analysis involves examining the context and content of digital evidence to determine relevancy. It also consists of following procedures such as statistical techniques and data mining in order to achieve results.

Presentation includes preparing documentation for reporting purposes.

Since the above phases share and involve common tasks, some of them could be grouped into a higher level; for instance, the identification, preservation, and collection phases could be grouped under one category called gathering. Furthermore, the examination and analysis phases could be grouped within a processing phase.

2.3.2 Proactive Model

Although reactive forensic investigation might lead to acceptable results in some scenarios, there are many cases in which an organisation could benefit from being able to record, gather and analyse intelligence proactively prior to tackling an incident. Furthermore, being proactive—also called digital forensics readiness—with regard to the design of the forensics capability within the organisation will ensure that the incident response team is able to respond effectively and efficiently (Clarke, 2010). There are several ways to implement such a model, one of which was introduced by Alharbi *et al.* (2011). They proposed a hybrid model of both reactive and proactive digital forensics that would operate simultaneously, in order to provide a framework that incorporated different forensics analysis algorithms to collect, trigger a malicious event, and preserve and analyse evidence proactively to identify an incident as it occurs, as Figure 2.2 illustrates.

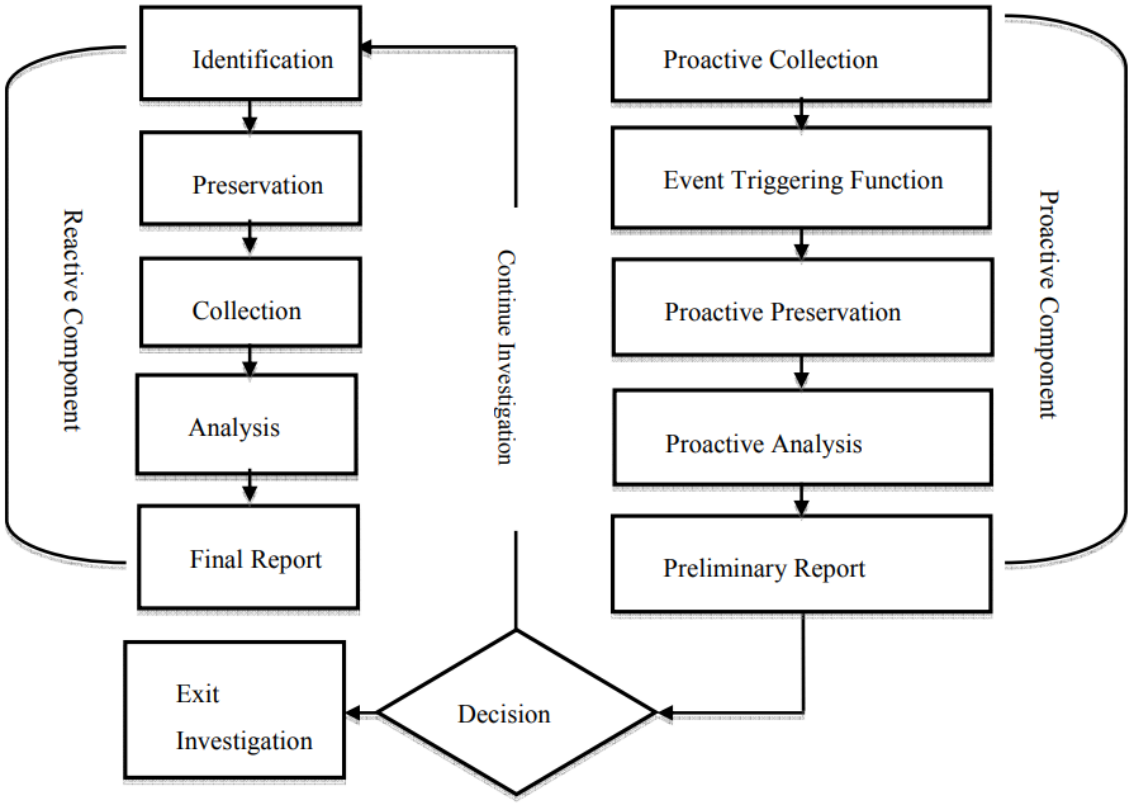

Figure 2.2: Proactive and reactive digital forensic investigation system process (Alharbi, Weber-Jahnke and Traore, 2011)

Several areas need to be addressed when an organisation intends to implement forensic readiness. These areas include technical actions, such as continuously monitoring and triggering responses to suspicious events. Non-technical dimensions are also involved in the process of applying proactive digital forensics effectively; for instance, staff training and conducting targeted security awareness programmes will genuinely boost the organisation's overall incident readiness. Moreover, it is necessary for the organisation's security policy and strategy to be a good fit with its digital forensic goals.

2.4 Challenges in Digital Forensics

Criminals are becoming more aware of digital forensics and investigation capabilities. As a result, they can defeat the process of investigation in some cases, which makes the analysis and reconstruction of attack scenarios more difficult and challenging (Rekhis and Boudriga, 2012). In addition, adversaries are even developing 'anti-forensic' methods and tools specifically designed to conceal their activities and destroy any remaining digital evidence. Privacy/ confidentiality-preserving techniques put in place, such as the integration of secure encryption into operating systems, are creating further challenges for forensic examiners; this has the potential to complicate the recovery of digital evidence from a computer, mobile or electronic device (Casey and Stellatos, 2008). Different encryption algorithms can be used for this task, which relies upon the preference of the system designer as to whether a secure encryption algorithm is required, such as the Advanced Encryption Standard (AES). Using a combination of a robust cryptographic algorithm and a high-entropy long key makes the process of brute-forcing the encryption key or breaking the algorithm encryption process a much harder task. This also slows the pace of digital investigation operations, as such a task takes an enormous amount of time to achieve (if it succeeds) and is sometimes unbreakable; it is here that the term 'anti-forensic' emerges. In the UK, the Regulation of Investigatory Powers Act (RIPA) 2000 states that it is illegal for a criminal not to reveal the access code (e.g., password or encryption key) to the law enforcement agency concerned. Failure to do so is considered a criminal offence, with a penalty of two years in jail (RIPA, 2014). It can be presumed that the law enforcement agency involved may

be able to build enough circumstantial evidence to prosecute. Nonetheless, according to Udham Singh (2014), around two-thirds of cases that involve encrypted files or devices go without processing, due to their inaccessibility even when using advanced forensic tools such as Forensic Toolkit (FTK) or EnCase Forensic Software (Eden *et al.*, 2016).

An interesting example that shows how encryption could pose an extreme challenge to law enforcement agencies when conducting investigations is the recent case involving the US Federal Bureau of Investigation (Apple, Inc. 2016). The FBI had asked Apple to make a new version of its most popular mobile device (iPhone) operating system (OS) to circumvent several essential security features, including the user's data encryption, and to install this new OS on the device so that the FBI could recover and have backdoor access to locked or encrypted mobile devices during investigations. Moreover, the use of biometric signals (e.g., face, fingerprint and iris) in encrypting files and for access control introduces further barriers to digital forensics investigators when attempting to bypass such constraints to complete an investigation process and uncover evidence (Abed *et al.*, 2019).

Other emerging domains in digital forensics that pose technological and legal issues are:

- Cloud computing forensics
- Big data forensics
- Internet of things forensics

The dynamic nature of cloud computing models includes the different services (software, platform and infrastructure as a service) that make it much harder for investigators to collect and analyse evidence. For instance, the same cloud infrastructure used by a criminal who conducted a massive and destructive cybercrime yesterday is being used today by another firm for a legitimate purpose. Some of the conventional computer forensic models and approaches are insufficient and incompatible with the cloud environment (Grispos, Storer and Glisson, 2012; Zawoad and Hasan, 2013).

One of the biggest challenges in investigating and prosecuting cybercrime is the Trojan/Virus defence. The Trojan defence is used when blaming an offence and *actus reus* (guilty act) on a piece of software, known as a Trojan (Brenner, Carrier and Henninger, 2004). It has now become common for individuals accused of a computer-related crime to claim that the responsibility lies with malware placed on their machine without their knowledge (Bowles and Hernandez-Castro, 2015). Several cases have ended with the release of suspects without being sentenced and walking free from court due to the Trojan/Virus defence (Fox News, 2009; Rutherford, 2010; Dalrymple, 2013). The current solution involves a layering of security countermeasures, which include the comprehensive logging of servers (including authentication requests) so that logs can be correlated in order to understand who was using which machine at what time that resulted in specific actions in a network (Quick and Choo, 2013; Ho, Kaarst-Brown and Benbasat, 2018; Khan, Foley and O'Sullivan, 2019). Assuming encryption is in place, proxy-based network decryption and storage of network traffic is required to identify misuse (possibly over prolonged periods of time). If third-party encryption is used,

it will not be possible to decrypt and inspect the traffic. An underlying assumption is that the computer account identified as being where the misuse occurred belongs to the individual who perpetrated the attack. However, with generally poor password use (e.g., shared and stolen accounts) and specific malicious intent, the assumption is unlikely to be true. Therefore, an approach that could biometrically correlate a user-based event/action directly with the individual who conducted it is required. Proactive digital forensics approaches—such as the method proposed in this research—could play a vital role in such cases and identify the real criminal by establishing a correlation between identity and evidence, such as a digital object.

Another major challenge that digital forensic investigators encounter when carrying out a deep inspection of an acquired forensic image is that information can be hidden inside image or text files. Hiding information is a well-known method of transferring knowledge and communicating over an untrusted channel that is used by criminals and spies to bypass monitored channels.

2.5 Digital Steganography

Steganography is the art of hiding information using a cover object, such as embedding a secret message inside a legitimate image, document, audio or video file, with the aim of not revealing a secret or leaving any signs that could indicate the hidden message, other than to the target recipient. In contrast, cryptography encrypts a message in such a way that it becomes unreadable by rendering it a meaningless jumble of characters, but it is present, and not hidden. The advantage of steganography over the use of cryptography alone is that the

intended secret message does not attract attention to itself as an object of scrutiny. However, for a long time, the difficulty of constructing the ordinary letter made steganography nearly going to be information modification. Traditional steganography methods, such as the least significant bit (LSB), which modifies the cover data to hide information, did not strictly satisfy the condition of being undetected and were vulnerable to being cracked by extensive steganalysis techniques (J. Liu *et al.*, 2018).

There is also a difference between steganography and a related data-hiding field called watermarking. Although steganography and watermarking have some fundamental similarities, in that they both hide secret information, they address distinct applications. In steganography, the carrier object (e.g., image) is a mere decoy and has no relationship with the secret message. In comparison, watermarking typically carries additional data about the carrier object or some other information related to the cover, such as tags classifying the source or the destination of the message being sent (Fridrich, 2009). In summary:

- Steganography is the process of sending a secret message by hiding it in a carrier object.
- Steganography is typically described as the prisoners' problem, in which two prisoners, Alice and Bob, want to hatch an escape plan but their communication is monitored by a watcher (Eve), who will halt the communication once she suspects an exchange of secret data.
- The most important property of steganography is that it is statistically undetectable, which means that it should be difficult for Eve to notice the

presence of a private message in a carrier. Statistically undetectable steganographic algorithms are referred to as secure.

- A warden who merely observes the traffic between Alice and Bob is referred to as passive. An active or malicious warden would tamper with the communication in order to prevent the prisoners from using steganography or to trick them into revealing their communication.
- Digital watermarking is a data-hiding application that is related to steganography but is fundamentally different. Whereas in steganography the secret message usually has no relationship with the cover object, which merely plays the role of decoy, watermarks typically supply additional information about the cover. Moreover, and most importantly, watermarks do not have to be embedded undetectably (Fridrich, 2009).

Figure 2.3 shows a typical steganographic system process flow that incorporates encryption when hiding secret data.

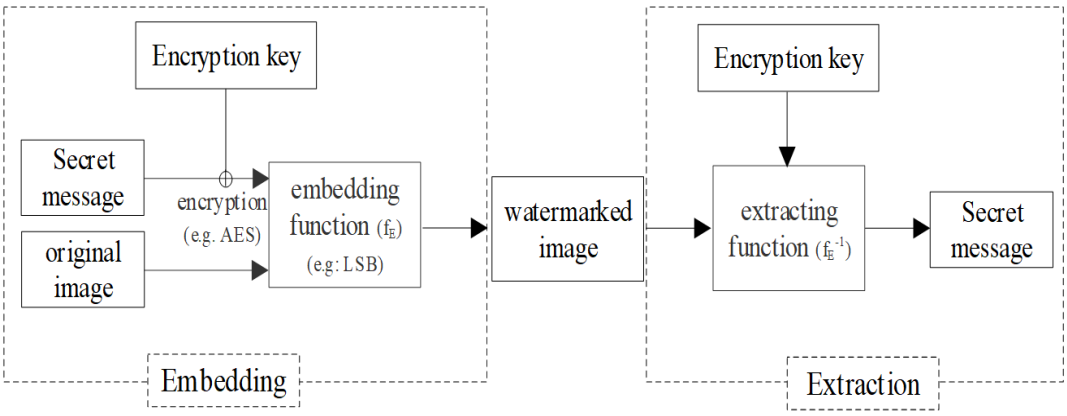

Figure 2.3: Typical steganographic system process flow

The steganographic process typically begins by encrypting a plain secret message using a symmetric encryption key, which is shared between the sender

and the receiver of the message to encrypt and decrypt the payload (the plain message to be embedded). Encryption is a typical practice in such applications; however, it is not always necessary. There are different techniques for hiding information in digital files. These techniques include four main methods: substitution, transform domain, statistical, and distortion.

2.5.1.1.1 Substitution System

Substitution-based techniques replace unnecessary or redundant bits of the carrier object with the secret message required to be hidden (Chan and Cheng, 2004). A well-known example of this approach is LSB-based steganography, as the LSB mostly has a low affect on the cover object, even when it is being removed or replaced with a different value (Chan and Cheng, 2004). A practical example of a substitution system with the LSB technique is replacing the last bit of each colour band of the pixels of an image with one bit of the secret massage (Wang, Lin and Lin, 2001). In the case of an RGB image, each pixel basically consists of three bytes: red, green and blue, and each pixel can accommodate three bits of the secret message (QL *et al.*, 2018). The secret message can be encrypted prior to being hidden within the cover object to preserve its secrecy in the case of someone detecting it. However, adding this type of security cover to a message introduces noise, as well as increasing the size of the original secret message. Another downside of the substitution system, in addition to its being naive, is that the secret message must almost always be smaller than the cover object (Wang, Lin and Lin, 2001; Chan and Cheng, 2004). OpenStego is an example of a software tool that utilises such an approach, in that it provides two

main functionalities: data hiding and watermarking (*OpenStego*, 2017). It can hide any data within a cover file (e.g., images). It can also watermark files with an invisible signature. This type of approach is useful for detecting unauthorised file copying and illegal distribution (Vaidya, 2019).

2.5.2 Transform Domain

In the transformation domain, the original space in the cover image is transformed into a different space, using, for example, discrete cosine transform (DCT), discrete Fourier transform, or wavelet transform (Poljicak, 2011; Cao *et al.*, 2018). These transformation techniques help in dividing the original image into parts based on their signal strength or importance, whereby those parts with low importance can be utilised to accommodate the secret message and reduce the possibility of its being visually detected. In the Fourier transform, the original image is transformed from the spatial domain into the frequency domain (Kipper, 2003). Once the cover object (i.e., image) is transformed into the frequency domain, the low signals are leveraged to hide the payload and the image is then transformed back to its original spatial domain (Cheddad *et al.*, 2010; Poljicak, 2011). Wavelet transform is very similar to Fourier transform but involves more complex mathematical operations (Edward Jero, Ramu and Ramakrishnan, 2014; Miri and Faez, 2018).

2.5.3 Statistical Method

Statistical methods modify the cover object by changing only a single bit, in which the modified object is considered as '1' and the unmodified object is considered as '0'. Therefore, the approach mainly relies upon the receiver detecting the

change and reconstructing the secret message by identifying the sequences of modified and unmodified objects to form the whole of the secret. OpenPuff is a statistical-based steganographic and watermarking tool that allows users to hide data in more than one carrier file (Sloan and Hernandez-Castro, 2018). When hidden data are divided between a set of carrier files, a carrier chain is created, with no enforced theoretical hidden data size limit (256MB, 512MB, depending only on the implementation) that implements three layers of concealed data obfuscation (cryptography, whitening and encoding) that extend deniable cryptography into deniable steganography.

2.5.4 Distortion Techniques

A distortion technique makes a sequenced change in the cover object in a way that it is only possible to decode the secret message by having both the original and the modified version of the cover and then performing a comparison operation to identify the changes. For example, Kim, Duric and Richards (2006) proposed an algorithm for hiding information in the LSBs of JPEG coefficients. The algorithm uses modified matrix encoding to choose the coefficients whose modifications introduce minimal embedding distortion. The expected value of the embedding distortion is derived as a function of the message length and the probability distribution of the JPEG quantisation errors of the cover images.

2.6 Digital Watermarking

Many books, audio and video publishers and distributors use digital watermarking to protect copyright and combat piracy. In simple terms, *digital watermarking* can be described as a method of encoding arbitrary information into digital objects,

such as using a unique customer ID number or company logo (Shih, 2017). Utilising digital watermarking could help in identifying the source of the illegal distribution of a watermarked digital object. For example, in the case of music piracy, when someone buys an album or song, the producer or distributor can leverage watermarking techniques to hide a unique identifier in the song file that could lead to the person who buys it and then trace the source of the illegal activity. Embedded digital watermarks can be used to verify the authenticity or integrity of the carrier signal legitimately or to show the identity of its owners. Figure 2.4 shows the system components and process flow of general high-level digital watermarking.

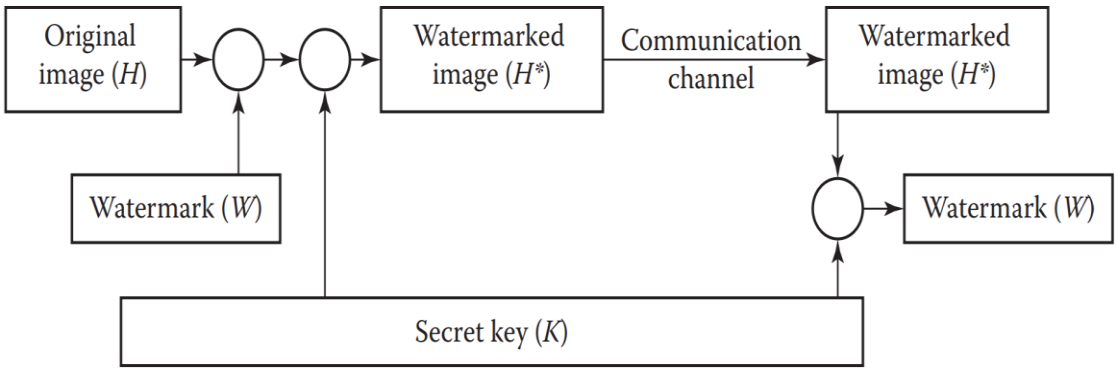

Figure 2.4: General digital watermarking system (Shih, 2017)

Overall, watermarking is very similar to a generic steganographic system. Hence, steganographic techniques can sometimes be leveraged to perform a watermarking task. Some digital watermarking schemes embed a unique identifier inside a digital object that needs to be protected in a way that is not visible, by using, for example, robust embedding algorithms such as those that

work in the spatial or frequency domain. This makes a malicious intent to remove the embedded watermarking information a non-trivial task to achieve.

2.7 Role of Biometric Technology in Digital Forensics

Biometric technology has various vital applications in forensic science (Gonzalez-Sosa *et al.*, 2018; Toor and Wechsler, 2018). Biometrics has contributed significantly to forensic investigations. For example, it has been used to identify missing individuals following natural disasters or accidents, such as fires. Moreover, biometrics can help in identifying attackers who are involved in acts of terrorism (Gonzalez-Sosa *et al.*, 2018; Toor and Wechsler, 2018). Some of the biometrics modalities and traits can be captured physically (e.g., fingerprints, palm prints, DNA), and others captured digitally or transparently (e.g., face, voice, body measurements, gait).

The use of facial pictures and video data in an investigation helps investigators to track suspects' faces, their locations, time of the crime, the people who are seen with them, and their activities (Carrier, 2003). However, uncooperative subjects, the low quality of the pictures and poor illumination make the recognition process a challenging task for any digital forensic facial recognition system. For example, for the Belgium Zaventem Airport attack in 2016, limitations in the forensic face recognition system had a serious impact on the ability to perform successful recognition, delaying the identification of the suspects in a time-critical investigation (Shoichet, Botelho and Berlinger, 2016). The system failed to track a suspect because of the distance between the camera and the individual, resulting in low resolution of the face, and variations in illumination. Different

expressions and facial poses can also cause a failure in facial recognition (Al-kawaz *et al.*, 2018).

Some attributes closely related to the human body, such as clothing and footwear, are often treated as biometric modalities in forensic science because they are collected, analysed and interpreted in the same way as biometric traces and exploited using the same inference models. The stability of a modality over time determines the obsolescence of the case-related data for investigation, from a lifetime for fingerprints and DNA, to some months or years for face and speaker recognition (Meuwly and Veldhuis, 2012).

In order to be of forensic interest, the biometric modality has to be available as a trace and needs to be distinctive. At a crime scene, fingermarks and biological traces are searched for as a matter of priority because they are often available and can be very distinctive. In comparison, iris pattern, although also very distinctive, is only rarely available as a digital trace. Digital traces may embed information about the body height of a perpetrator, but this modality is only privileged if no other option is available, due to its poor distinctiveness. The modality also needs to be stable and robust in forensic conditions. Face recognition is commonly exploited for forensic investigation but suffers from severe limitations. For example, facial features can change significantly over even short periods of time and unconstrained video captures constituting the main part of the trace material can lose a substantial part of the distinctiveness of the modality (Meuwly and Veldhuis, 2012).

Unfortunately, as the use of technology increases, so, too, does digital crime and terrorism develop ways to avoid being discovered. Nevertheless, the increased use of technology could also provide an opportunity to derive new biometric and digital signatures to pursue those who engage in criminal and terrorist activities, as some electronic devices have their own unique digital signature (Jain, Flynn and Ross, 2008). Human interaction with electronic devices also provides some interesting new digital biometrics (Iyengar and Miller, 2015).

There are several open-source and commercial facial recognition systems that can be leveraged for use as a digital forensic tool to aid in identifying and matching individual suspects (e.g., AmazonRekognition, 2015; Baltrusaitis, Robinson and Morency, 2016; Clarifai, 2018; Neurotechnology, 2018; GoogleCloudVision, 2019; MicrosoftFace, 2019; OpenCV, 2019). Such systems' accurate search capability allows investigators to identify (to a certain degree) a person in a photo or video, using a private repository of face images. Figure 2.5 illustrates the performance of three of the commercial systems--Neurotechnology, Microsoft, and Amazon's Rekognition—as they were examined in identifying individual faces based on a public dataset called CAS-PEAL-R1 (Wen Gao *et al.*, 2008). The dataset consisted of 30,900 images across 1,040 subjects (595 men and 445 women). A study by Al-kawaz *et al.* (2018) included only those images that met all conditions (pose, illumination, expression, and accessories) (95 subjects); the remaining were excluded in order to investigate the performance of the three commercial systems in different environmentally realistic conditions.

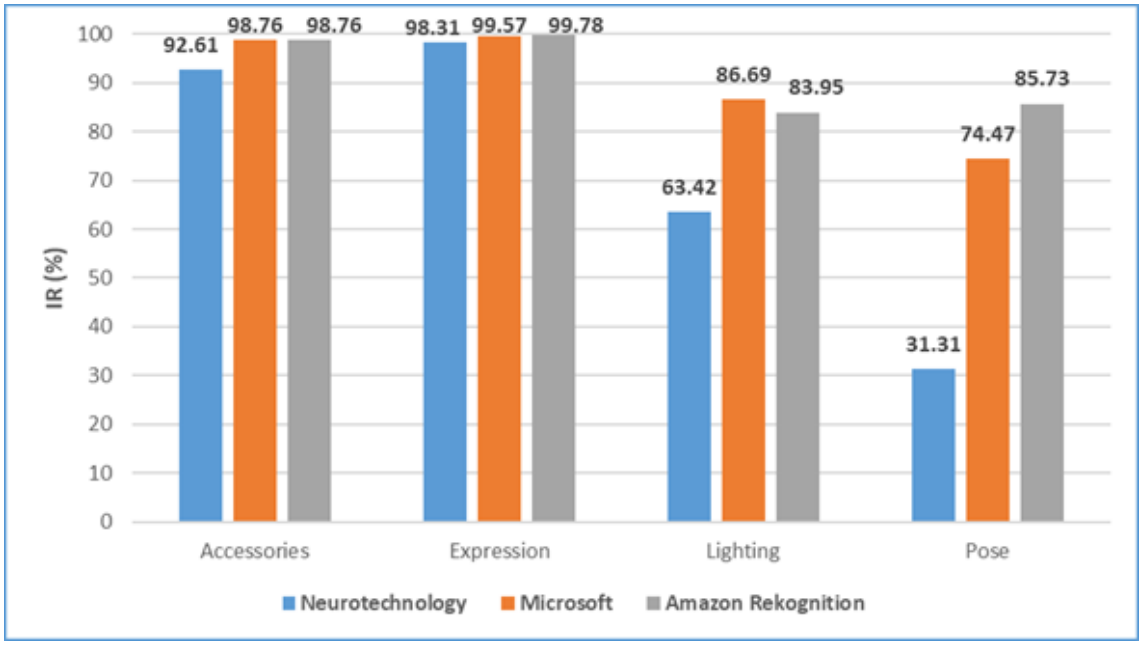

Figure 2.5: Performance of three commercial facial recognition systems using the CAS-PEAL-R1 dataset (Al-kawaz *et al.*, 2018)

It can be seen in the above figure that the systems examined mostly achieved a more than 80% success rate in matching a given face image with the person to whom it belonged. However, in some conditions, such as different lighting variability and pose orientation, the performance was much lower. It is important to mention that the subjects' pictures that were examined were captured using a professional digital camera, in which the image resolution is high and all the faces fully presented. More challenging settings could include a situation in which only part of the person's face can be seen. Identifying someone from partial, unclear details of a face could be extremely challenging.

2.8 Conclusion

This chapter presented the concerns regarding threats from insider misuse along with the existing challenges and barriers that face a typical reactive digital forensics investigation. Forensic practitioners encounter complex challenges, including data encryption, information hiding and data destruction. Although much research has been undertaken into the detection of insider misuse, and particularly information leakage, less focus has been given to linking this to the people who are committing the criminal activity. Assuming many will utilise credentials stolen from colleagues to perpetrate misuse, there is frequently a reliance upon other security controls to verify the identity of the person responsible (e.g., through the use of logs, CCTV, etc.) if indeed this is possible at all. As a result, much more is required from specialists in the domain in terms of incorporating different technologies, such as biometrics, into forensic readiness and ensuring that incident response teams and forensic investigators are able to respond effectively and efficiently once an incident occurs. Therefore, aligning individuals with the data objects with which they are interacting and leveraging biometric signals and systems and steganographic techniques could be a useful combination to attribute misuse and enable digital forensic investigations to reach positive results.

3 Data Loss Detection, Prevention and Proactive Digital Forensics

In order to provide a better understanding of the existing literature, this chapter provides a review of relevant research and state of the art on data loss detection, prevention and proactive digital forensic techniques. The chapter begins by describing the methodology used for conducting this review, followed by a detailed critical review of related studies. The chapter concludes with a discussion section that identifies a gap in the literature.

3.1 Introduction

A systematic literature review is used in this chapter as the review methodology to identify, evaluate and interpret all published research relevant to the topic area. This method follows a specific protocol that makes the result reproducible. Applying a systematic review involves three main steps: planning, execution, and documentation of the results, as illustrated in Figure 3.1 (Harris *et al.*, 2014; Xiao and Watson, 2019). Each step includes several sub-steps that ensure the consistency of the search activity.

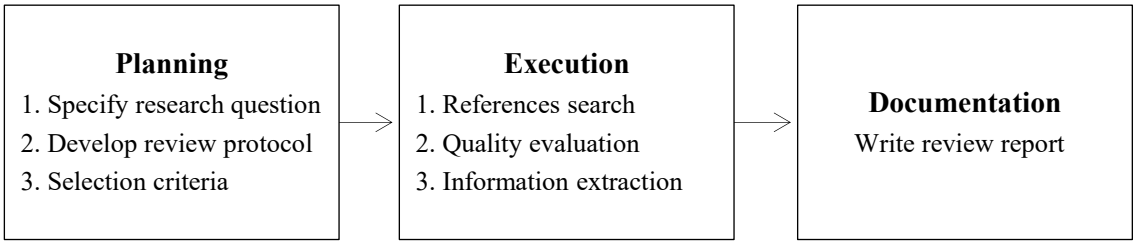

Figure 3.1: Systematic review methodology

In the planning phase, the author identified research questions the review should answer. Specifying the research questions was the most important activity, as the questions drove the entire literature review. The main research question selected is: “What are all the existing data loss detection, prevention and proactive digital forensics solutions?” The review was conducted from an academic perspective. The next section explains in detail how the systematic literature review was undertaken.

3.2 Literature Review Methodology

In addition to the main research question referred to above, the following sub-questions helped to meet the established goals:

- What are the techniques used to detect data leakage?
- What are the document classification methods?
- What are the existing techniques for the identification of guilt?
- What are the existing proactive digital forensic techniques?
- What measures are used to evaluate performance?
- What was the performance achieved?

A review protocol was also developed to describe how the collected data were selected. A protocol establishes predefined methods that reduce the possibility of researcher bias (Budgen and Brereton, 2006). For instance, instead of selecting review papers based upon the researcher’s expectations (in the absence of a predefined protocol), the selection is driven systematically when a protocol is used. The review protocol is explained in detail in the next section. During the

planning phase, selection criteria are specified to identify original papers that are directly related to the research question. Therefore, the criteria are used to include as many relevant studies as possible and exclude irrelevant material. Five filters were applied to the search results returned in order to identify the most relevant studies:

1. Papers with titles that clearly show no link to the primary question were excluded from the review. For instance, many of the titles in the search results have the words “water leakage”, “CO2”, and so forth.

2. Publications of less than two pages (including posters, presentations, abstracts or some short theoretical papers) were excluded.

3. Non-peer-reviewed publications were eliminated.

4. No studies published prior to 2007 were included, the researcher has conducted a preliminary search that resulted in no significant or up-to-date research existing before that date.

5. The literature review focused only on those publications written in the English language.

The systematic literature review was executed in three steps: references search, quality evaluation and information extraction. It is essential to mention that the search activity was limited to those academic papers that were indexed by reference databases available on the Internet. After the research questions were specified, four main research repositories were listed as being the most relevant to the field of the literature domain. Google Scholar—a public index database—was

used as a sanity check. Various carefully chosen search terms and expressions relating to the main questions were used during the enumeration activity. The following databases were considered in the review:

1. IEEE Xplore: http://ieeexplore.ieee.org/Xplore/home.jsp
2. ScienceDirect: http://www.sciencedirect.com/
3. ACM Digital Library: http://dl.acm.org/
4. SpringerLink: http://link.springer.com/
5. Google Scholar: http://scholar.google.co.uk/

After performing several search tests on the above databases, the following composite search expression was formed for the enumeration:

"(Data OR Information OR Document OR Documents) AND (Loss OR Leak OR Leakage) AND (Prevention OR Detection)"

The use of "OR" and "AND" together facilitated the automation of 24 possible phrase combinations.

Table 3.1 lists the number of publications returned for each database after several enumeration rounds using the above query. The table also presents the final number of studies selected after applying the inclusion/exclusion criteria.

Table 3.1: Number of returned references

Database	Number of references	Final selected references
IEEE Xplore	58	10
ScienceDirect	18	4
ACM Digital Library	11	2
SpringerLink	9	3
Google Scholar	421	15
Total	517	33

In addition to the selection criteria, the primary publications were classified based on the quality of each study, as Table 3.2 illustrates.

Table 3.2: Publication genre and the number of primary studies

Genre	Number of primary studies
Peer-reviewed journal papers	16
Peer-refereed book chapters	2
Peer-reviewed conference papers	15
Total	33

While reviewing the research papers, the following set of information was extracted from each study:

- Experimental methodology

- Dataset information
- Evaluation of performance
- Contribution of the study
- Study limitations

A comprehensive critique and interpretation of the primary studies is presented in the next section.

3.3 Literature Review of Data Loss Detection, Prevention and Proactive Digital Forensics

As part of the literature review, the publications were carefully classified into seven main groups based upon the objective each publication was trying to achieve. These groups were:

- Document classification
- Data loss detection in email
- Data encryption and access control
- Biometrics and human behaviour
- Distributed and host-based solutions
- Guilt identification
- Proactive digital forensics

It is noteworthy that the reviewed studies overlap in several aspects, such as the methodology used for detecting leakage (such as a classifier algorithm) or the type of action applied once a leak is detected.

Shabtai *et al*. (2012) proposed a taxonomy of existing data loss detection and prevention (DLD/P) solutions, as shown in Figure 3.2, which incorporates data state, deployment scheme, leakage handling approach, and action was taken upon leakage. These attributes answer the questions of what, where, and how to protect sensitive data. The taxonomy covers a wide range of aspects and technologies of DLD/P. The reason the organisation of this literature review does not precisely follow the structure of the Shabtai *et al*. (2012) taxonomy is that most of the studies investigated here contribute to a specific aspect of DLD/P, whereas others fall into multiple categories. For instance, some publications propose a classification technique to cluster documents based on context, and other studies focus on email patterns using machine learning. Therefore, it was not feasible to categorise the literature using Shabtai *et al.* (2012) classification. As a result, the reviewed papers are mainly grouped based upon the aim of the study and the methodology used.

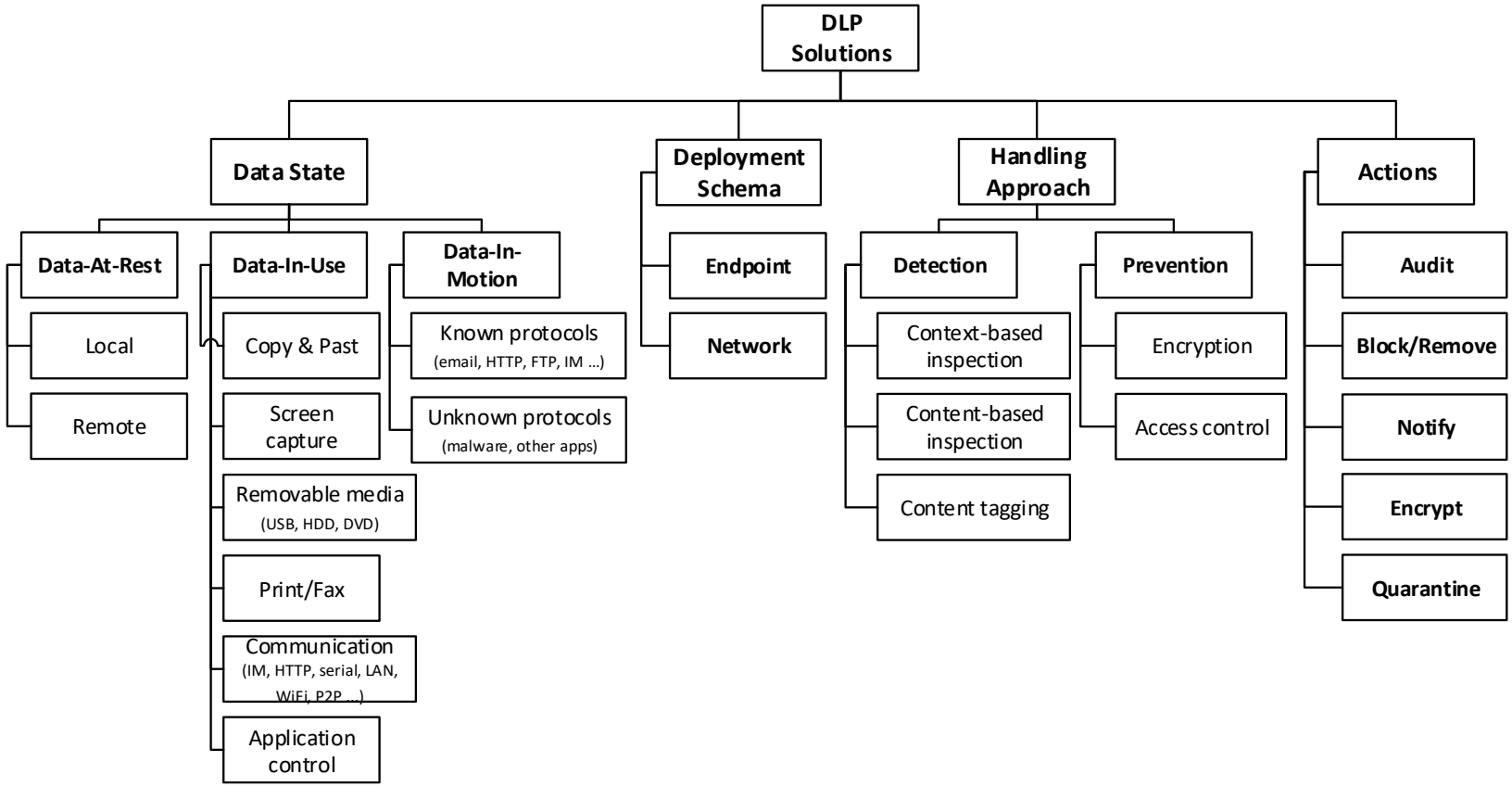

Figure 3.2: Taxonomy of data leakage prevention solutions (Shabtai, Elovici and Rokach, 2012b)

3.3.1 Document Classification

Document classification or categorisation is an active area in which much research has been undertaken and many studies have discussed in detail the clustering of data for DLD/P. A study by Gessiou *et al*. (2011) proposed an information retrieval-based method for information leakage detection (IRILD). Fundamentally, their technique eliminates common phrases—by checking their popularity—before performing a fingerprinting process. This results in only sensitive phrases being fingerprinted, leading to much faster processing time with fewer fingerprints generated. The fingerprint process uses a cyclical hashing approach, which splits the document into multiple parts and generates fingerprints for those parts. The proposed method provides better performance, as it generates fewer fingerprints and reports fewer false positives (as illustrated in Figure 3.3).

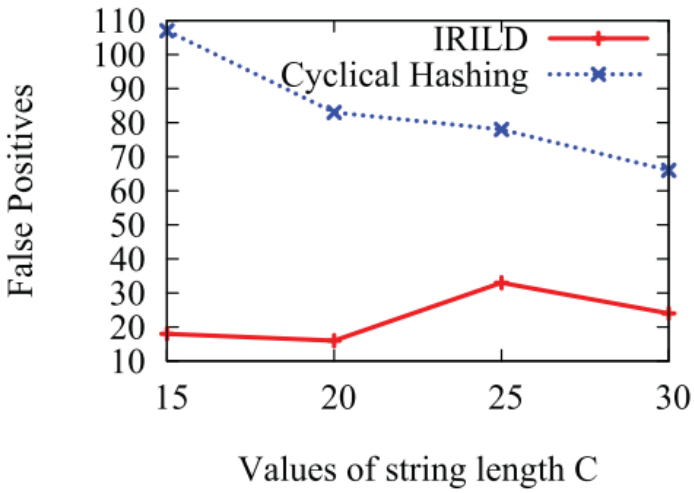

Figure 3.3: Accuracy of varying the length of C (Gessiou, Vu and Ioannidis, 2011)

Gessiou, Vu and Ioannidis (2011) conducted their experiment on an Enron email dataset that contained around 600,000 emails. IRILD managed to generate 39% fewer fingerprints compared with standard cyclical hashing. Unlike other research, this method does not rely on a predefined list of keywords given by the user. Conversely, the method is able to identify and classify confidential documents using the fingerprints it has generated. Thus, the system requires the use of public search engines, such as Google, Bing or Yahoo, to determine the common phrase score. In confidential environments, submitting such sensitive information to a third party could introduce a considerable risk to privacy. Furthermore, using phrase or expression popularity to measure document sensitivity is not always an accurate method, since not knowing the context in which the phrases appear affects classification accuracy.

Similar to the concept of the elimination of phrases introduced by Gessiou *et al*. (2011), Shapira *et al*. (2013) proposed an approach that distinguishes between non-relevant (non-confidential) and relevant (sensitive) sections of a given document. Their solution comes as an extension to the common fingerprinting approach, except that it uses k-skip-n-grams. The proposed method employs different "k" values to skip n-grams, which enables the system to conduct contextual analysis. This approach was more robust against content manipulation activities, such as rewording, paraphrasing, word deletion, and changing word order. Evaluation of the proposed method shows that the sorted k-skip-n-grams achieved higher detection accuracy than the traditional fingerprinting approach.

Likewise, Alneyadi *et al*. (2014) investigated the effectiveness of using n-gram-based statistical analysis for the purpose of document classification. The proposed method utilises n-grams to measure the frequency of a given word within a document. In this experiment, the system achieved an overall classification accuracy of 92% and scored an average recall of 91%, with 92% precision. Precision gives the fraction of the system prediction that is true; recall is the fraction of the relevant result that is successfully retrieved or detected (Davis and Goadrich, 2006).

Equation 1: Illustrates how precision is calculated using true positive and false positive values

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Equation 2: Illustrates how the recall is calculated using true positive and false negative values

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Although the system is able to attain a high level of accuracy in detecting sensitive documents, the length of a document (especially minimal and very large documents) can have a significant adverse effect on the accuracy of the classification. This is due to the proposed technique depending upon the frequency of a phrase, and frequency can be directly affected by the size of the document.

Katz *et al.* (2014) proposed an approach that leverages the advantages of both keyword-based and statistical methods. This approach consists of two phases: learning and detection. The learning phase generates a context-based confidential terms graph for all the types of the confidential document an organisation possesses. In the detection phase, the system calculates the confidentiality score by matching the document concerned to one or more clusters which have been identified in the learning phase, as Figure 3.4 illustrates.

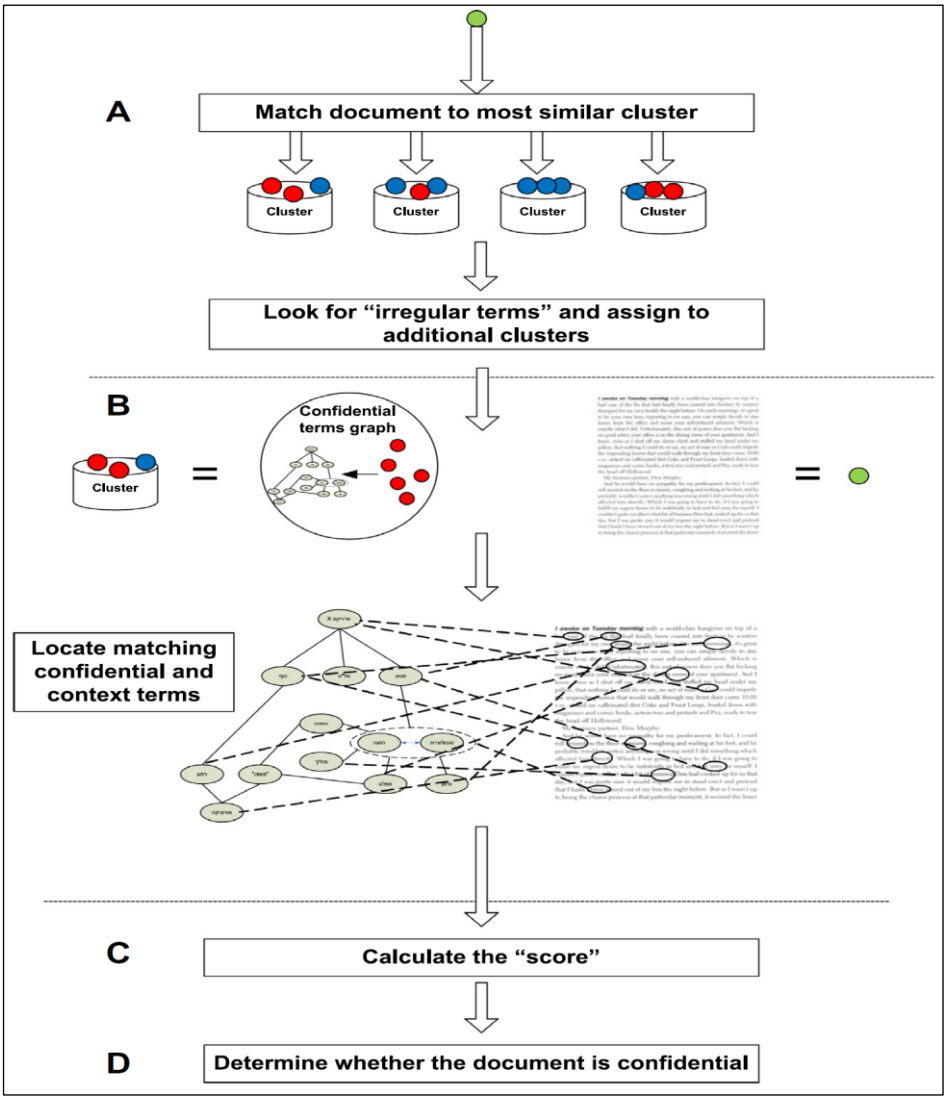

Figure 3.4: Detection phase (Katz *et al.*, 2014)

The system does not simply rely upon content or predefined keywords, but also takes context into account, as does the Shapira *et al.* (2013) approach, both of which are robust against rephrasing attacks. The system also detects small sections of confidential information embedded in non-confidential documents. The experiment was conducted on three datasets: Reuters news articles, the Pan-PC-11 Plagiarism dataset (containing 22,186 documents), and the Enron emails dataset. The authors collected a total of 6,102 news articles from Reuters. The result was compared with other methods, such as support vector machines (SVM), and fingerprinting. In terms of identifying whole confidential documents, all the methods scored very close true positive and false-positive rates of ~95% and ~10%, respectively. When detecting confidential sections in a non-confidential document scenario, the fingerprint algorithm achieved a higher true positive score than the proposed technique, followed by the SVM (90%, 70%, and 15%, respectively), all with ~5% false positives. It is clear that this system does not perform better than fingerprinting in terms of the detection rate in both scenarios. However, the authors claimed that the system addressed a new scenario: rephrasing small parts of a confidential text. It can be argued that other techniques had already investigated this particular aspect and managed to achieve a promising result, such as Shapira *et al.* (2013).

Another study that tried to overcome the disadvantages of a regular expression method was conducted by Du *et al.* (2015). The study introduced an approach that extracts a limited number of critical semantic features and necessitates a small training set. The ultimate objective of this study was to determine whether a given file belonged to the same class as the training set. The proposed method

uses latent semantic analysis (LSA), which maps documents to a vector space with reduced dimensionality. For feature extraction, LSA uses singular value decomposition (SVD), which abstracts the semantics of training documents and uses an algorithm to calculate those features. Given a training set of documents, its term- frequency matrix *A* cab be can be written as A = [ai,j] ∈ $R^{m \times n}$, where *m* is the number of terms, *n* is the number of documents and $a_{i,j}$ is the number of times term *i* occurs in document *j*. Given *A*, we use SVD to decompose the term-frequency matrix into three matrices:

$$A = USV^T$$

Where $S = diag(\sigma_1, \ldots, \sigma_{min(m,n)})$, $\sigma_1 \geq \ldots \geq \sigma_{min(m,n)}$ is a rectangular diagonal matrix. Each diagonal element in S corresponds to a concept or a semantic feature, as shown in Figure 3.5.

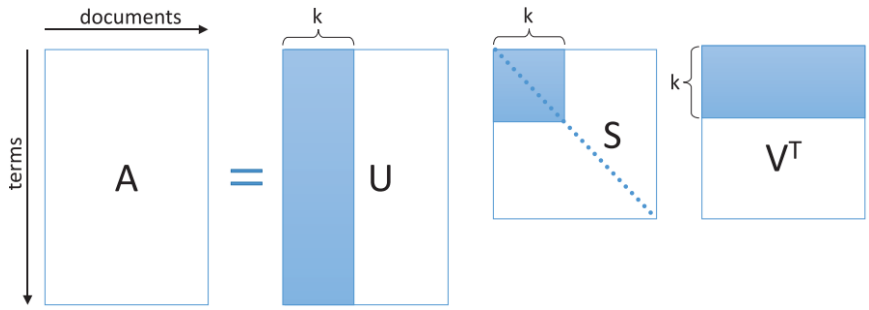

Figure 3.5: Singular value decomposition. Du *et al*. (2015)

In their experiment, Du *et al*. (2015) used documents on different topics, each category containing between 16 and 118 documents. They achieved true positive rates of between 76.1% and 98.6% and false-positive rates of between 0.8% and

15.1% for all document sets. A possible pitfall of their study is that the document dataset size could be considered small since other studies in the same field have used many more documents. For instance, Katz *et al*. (2014) used more than 6,000 documents to evaluate the performance of their proposed system.

3.3.2 Data Loss Detection in Emails

One of the most common data loss exit points is email. A considerable amount of research has been undertaken in detecting sensitive information sent by email messages, including email content inspection and behaviour-based analyses. For instance, Kalyan and Chandrasekaran (2007) proposed 17 variables as generic attributes that would encapsulate a sender's intent. The attributes include time slot, attachment type and size, mail body size, type of action on the email message (e.g., reply, forward, etc.), number of email addresses sent to, and the messages sent. The system analyses the value of the input variables by using a machine learning algorithm, and both SVM and neural networks were used in the experiments. To perform binary classification sing SVM, given training data (x_i, y_i) for $i = 1 \ldots N$, with $x_i \in R^d$ and $y_i \in \{-1, 1\}$, learn a classifier $f(x)$ such that

$$f(x_i) = \begin{cases} \geq 0, & y_i = +1 \\ < 0, & y_i = -1 \end{cases}$$

i.e. $y_i f(xi) > 0$ for a correct classification.

They evaluated their approach using a dataset that contained 554 emails, with 70 emails marked as leaked messages. The authors did not provide information about the dataset and whether it was real or artificially created for the experiment

purpose. The results show that the high performance of this method was demonstrated by achieving an overall accuracy of 92%.

Interestingly, the above results were obtained without performing any content inspection or relying on predefined keywords. However, a possible weakness of this approach is the necessity for user involvement in some of the stages. For instance, the system user must perform manual inspections to distinguish whether email messages are official or personal. Furthermore, the metric used for examining system performance can be strongly criticised, as accuracy is a vague metric for measuring the quality of such systems. Metrics such as true positives and false positives would more clearly demonstrate the performance of a method. Furthermore, the result cannot be compared with other studies in the same domain, since the source of the dataset is not defined.

Along the same lines, Liu *et al*. (2014) introduced features that can distinctly describe the characteristics of misdirected emails while preserving users' privacy. Misdirected emails are detected based on 22 features, none of which are content-based attributes. The researchers employed a random forest classifier to detect misdirected emails. The system evaluation used the version of the Enron dataset compiled by (Shetty and Adibi., 2004). Since the Enron dataset does not contain what the authors call misdirected emails, sample misdirected emails were generated by Liu *et al*. (2014) for each email in the training set for experimental purposes. Overall system performance reached an average recall rate of 82% with a precision rate of 89%.

Although both Kalyan and Chandrasekaran (2007) and Liu *et al.* (2014) used machine learning techniques, together with a set of generic variables as inputs to the classifiers used, it is not possible to compare the performance of the two studies. Different datasets and evaluation metrics were used in each piece of research.

A study by Manmadhan *et al.* (2014) used a naïve Bayes classifier to investigate the possibility of categorising sensitive emails effectively. Naive Bayes algorithm that uses conditional probability approach to classification is adopted (Manmadhan *et al.*, 2014b). The system architecture design consists of three engines that work in sequence: the pre-processing engine performs data cleaning by removing stop words and stemming as shown in Figure 3.6.

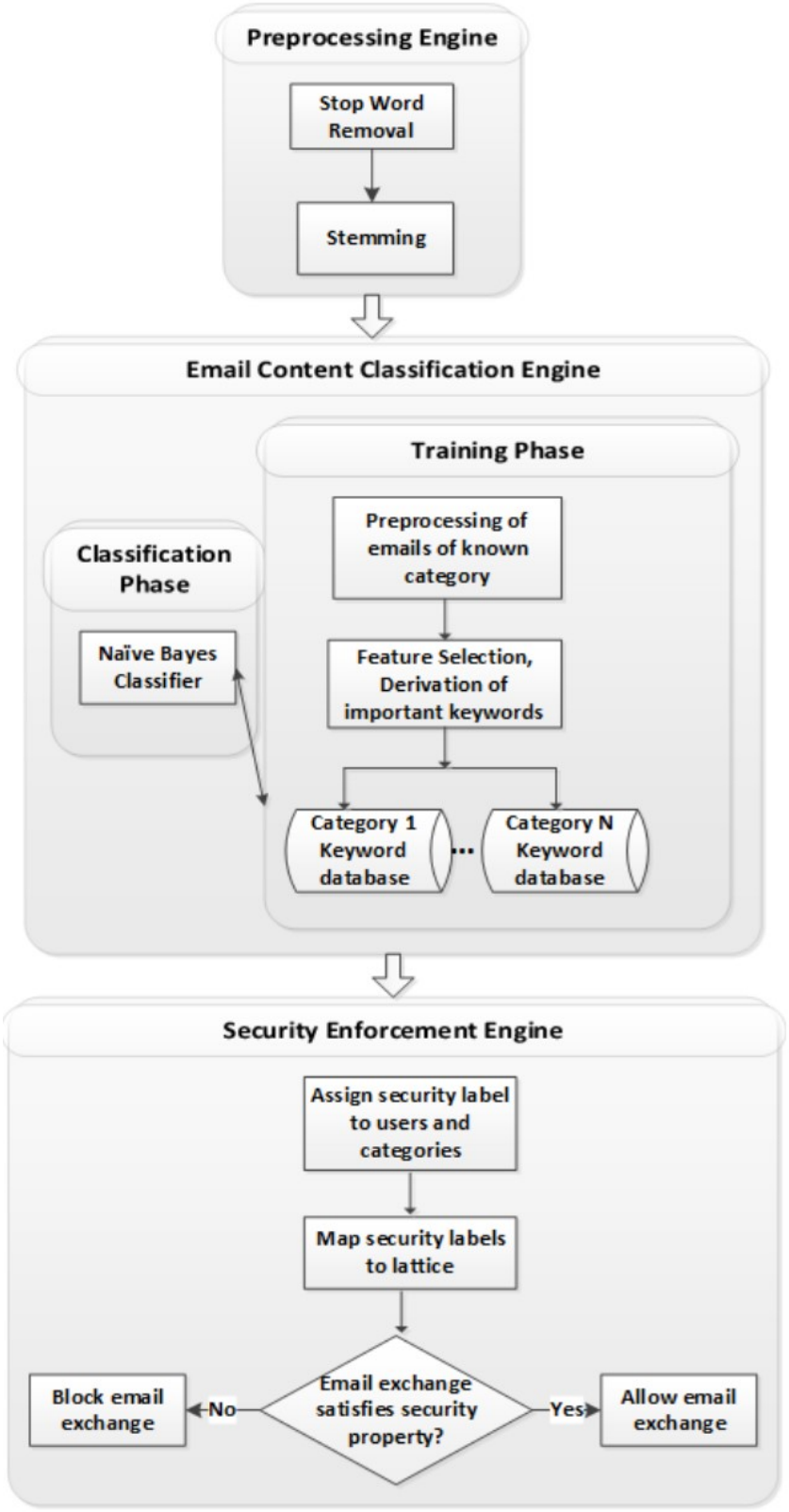

Figure 3.6: Architectural diagram (Manmadhan *et al.*, 2014b)

The output of the pre-processing engine is then fed into the classification engine using a naïve Bayes algorithm; finally, the security enforcement engine evaluates whether the email exchange satisfies the security property by applying a lattice model (Denning, 1976). For detection, the system uses predefined keywords to classify the documents into previously defined categories. As with Liu *et al.* (2014), the experiment used the Enron email dataset. The authors achieved an overall accuracy of 72.4%. Furthermore, what makes the Manmadhan *et al.* (2014) approach different from that of both Liu *et al.* (2014) and Kalyan and Chandrasekaran (2007) is that it performs content inspection, whereas the other studies used behaviour-based approaches.

Balinsky *et al.* (2011) attempted to overcome the loss of data via email by intercepting the corresponding application system calls. The general architecture of the framework is demonstrated in Figure 3.7. The system first automatically intercepts system calls, then performs a content inspection based on certain predefined keywords (e.g., “customer”, “agreement”, “contract”, etc.). After that, the system implements an appropriate policy based on the sensitivity level determined in the previous step. In contrast with other approaches which are limited to UNIX-based operating systems, this method works on Microsoft Windows operating systems, despite the complexity of intercepting its system calls. The policy framework used also covers a variety of leak points, such as peripheral devices, shared locations, and accidental email leakage.

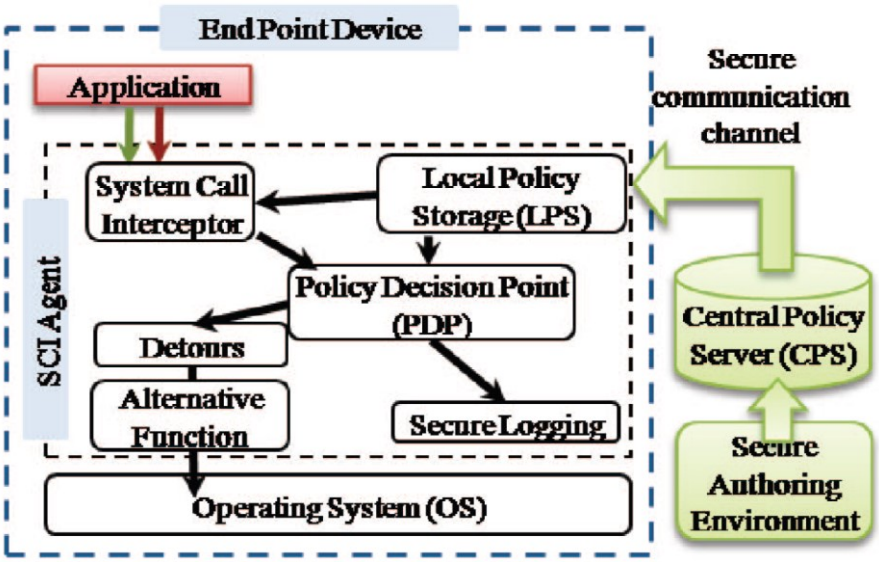

Figure 3.7: General architecture of a proposed framework (Balinsky, Perez and Simske, 2011)

The experimental results show that the overall process rate was 40-130 MB/ms. However, the evaluation of the system is ambiguous, since there is no information on how the system performs in terms of other metrics, for instance, true positives, false positives rates, f-score, precision or recall. Moreover, it seems that this system does not detect data leakage when a confidential message is embedded in a cover file, such as a text file embedded in OOXML files using the unknown parts and relationship data-hiding approach introduced by Park *et al*. (2009).

3.3.3 Biometrics and Human Behaviour

Some recent studies, such as Lee *et al.* (2013) and Lee *et al.* (2014), have researched an interesting approach to preventing data leakage by measuring the change in an individual's bioelectrical signals. Both studies tried to predict the intention of a data leak by analysing the change in the subject's biometric pattern, regardless of the identity of the subject in question. The method proposed by Lee

et al. (2013) also undertakes a proactive detection technique to measure the average value of an insider's biometric signal, such as pulse and skin conduction and compares it with one stored in a dataset. If this value exceeds the standard signal value, the system assumes that this is a sign of intentional data leakage. Nevertheless, the effectiveness of the method is arguable, since it is neither demonstrated nor explained how the technique assumes that the insider's abnormal behaviour is related to data leakage. The researchers also presented the basic design structure of their technique without conducting an evaluation or demonstrating any pilot experiments.

Later, Lee *et al.* (2014) published another study, in which they measured a subject's vital signs, such as heart rate variability, core body temperature, and skin temperature, while the subject watched a horror movie. The experiment showed that there was a noticeable change in the heart rate (HR) of the test subject when a dreadful scene suddenly took place during the movie. Despite this change in HR, the proposed technique is strongly questionable since the authors failed to indicate how the method could be transferred and implemented in the field of DLD.

Using another technique, Wu *et al.* (2013) investigated profiling a subject's keystroke behaviour along with discovering the input of sensitive data simultaneously. The system first hooks into the keyboard API records all the user's inputs and then eliminates irrelevant keystrokes and function keys, such as Enter, Backspace and Shift. The system then performs intensive analysis on the captured data and stores the results in a file that is compatible with the data

loss prevention (DLP) software (called Privacy ID) the authors used for a detection-free DLP solution that supports the Chinese language (AmXecure, 2013). Since the system detects sensitive information as it is being written, this eliminates the need for decoding or parsing different file formats. In terms of identifying the user, the system collects the time cost of characters typed by the user. It then uses an SVM to create the typing model of that user. The concept behind the proposed DLP model matching is presented in Figure 3.8.

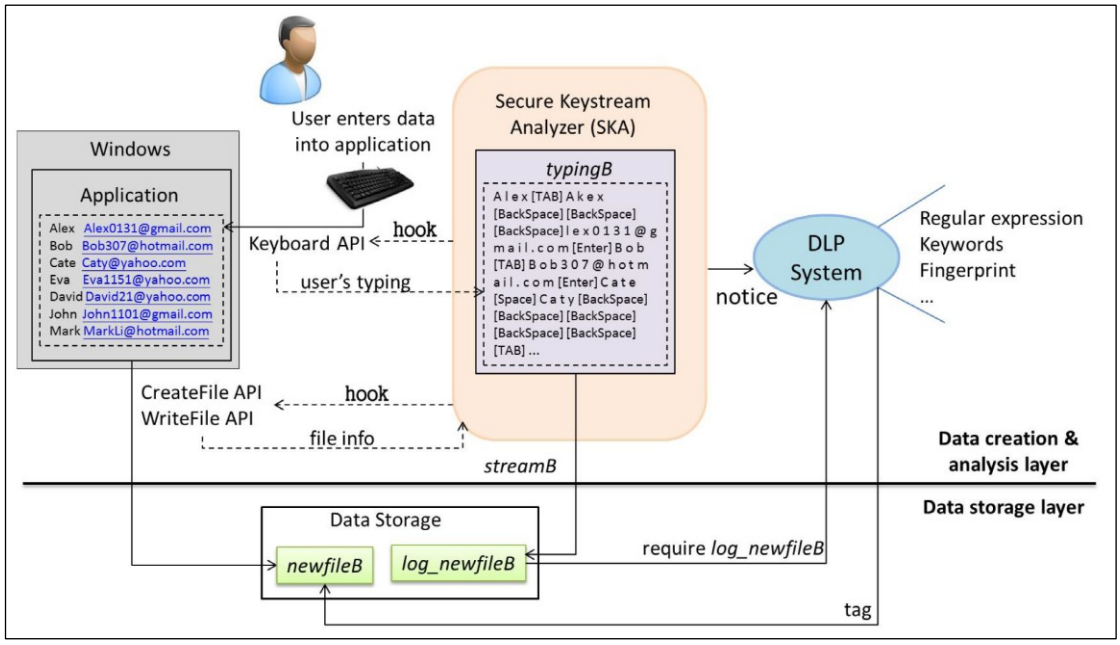

Figure 3.8: Concept behind a proposed DLP model matching (Wu *et al.*, 2013)

The method of keystroke behaviour was able to score an average accuracy of ~80%. Overall, the experiments conducted demonstrate that this system positively employs keystroke profiling for detecting data leakage and dynamically identifying malicious intent. The system also resolves the issue of parsing different file formats that are encountered in current commercial DLP systems, since it does not require file inspection or content analysis. However, confidential

data could be leaked without being detected by the system as easily as sending a document via email, since the proposed system only intercepts data input via the keyboard. The proposed method is also unable to detect a simple copy and paste action, a shortcoming that could be used to manipulate data and avoid detection by the system. Furthermore, the system only detects keywords predefined as sensitive, whereas an insider could easily rephrase or misspell the input phrases to evade detection.

3.3.4 Distributed and Host-Based Solutions

Designing and evaluating a practical, working DLP solution is not an easy task. It involves several techniques that are required to work together, intelligently to detect and prevent leakage attempts. For instance, Lee *et al.* (2009) proposed an agent-based distributed system designed in a way that it separates the powers and responsibilities between the system's agents. The system consists of seven agents (as shown in Figure 3.9), three of which are installed on target hosts to perform tasks such as content inspection and watermarking.

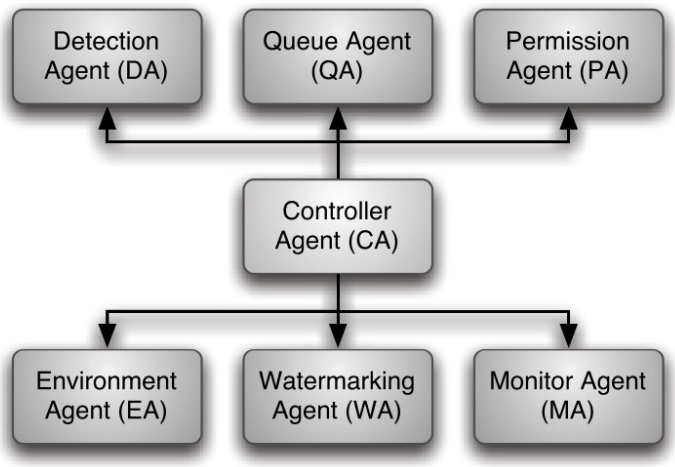

Figure 3.9: Agent classifications and hierarchy (Lee *et al.*, 2009).

The other parts of the system are centrally managed to control the ones that are distributed. Image watermarking (Dugad's algorithm) is used to embed permissions into the object content in a way that is ideally irremovable without rendering the file content (Dugad, Ratakonda and Ahuja, 1998). Dugad's algorithm adds the watermark to the significant coefficients in the DWT and it does not require the original image for watermark detection is presented.

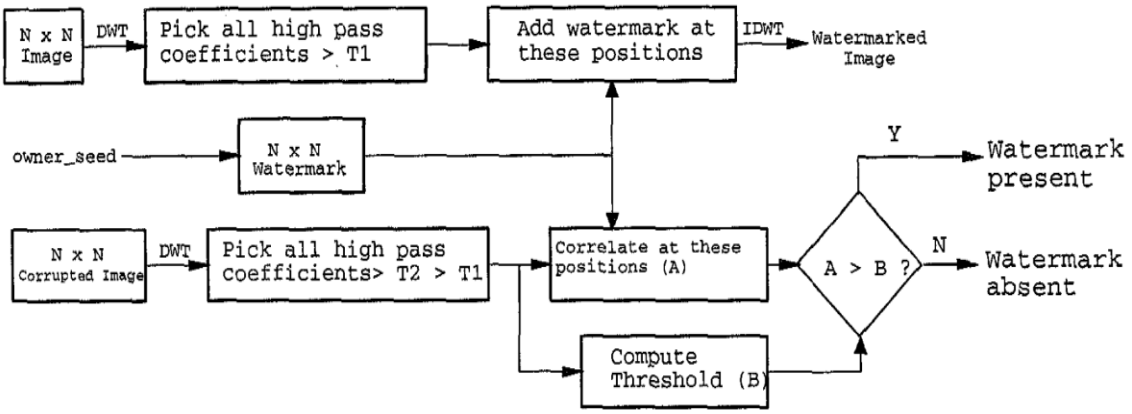

Figure 3.10: Dugad's image watermarking algorithm, the top part shows watermark casting and the bottom part shows watermark detection (Dugad, Ratakonda and Ahuja, 1998).

The following equations are using for the watermarking casting:

$$V'_i = V_i + \alpha \mid V_i \mid x_i$$

Where i runs overall DWT coefficients > T_1 (barring the low pass component). Vdenotes the corresponding DWT coefficient of the original image and V' denotes the DWT coefficient of the watermarked image. x_i is the watermark value at the position of V'. x_i is generated from a uniform distribution of zero mean and unit variance. α is taken as 0.2.

$$z = \frac{1}{M}\sum_{i} \tilde{v}_i \, y_i$$

Where i runs overall coefficients > T_2 > T_1 and M is the number of such coefficients. The threshold S is defined as

$$S = \frac{\alpha}{2M}\sum_{i} | \tilde{v}_i |$$

Despite the system's scalability and architectural complexity of the proposed system by Lee et al. (2009), the experiments focused only on image files, whereas most data loss incidents occur in the form of documents. Moreover, the paper does not include an evaluation that shows the system's performance. Nevertheless, the authors claim that the proposed system correctly identified all invalid watermark images and quarantined the modified ones.

From distributed systems to host-based deployment aiming to detect inadvertent information leaks, both Kemerlis *et al.* (2010) and Ko *et al.* (2014) have presented end-user solutions that are capable of overcoming the limitations of controlling suspicious events. These systems are designed for personal use only, in contrast with Lee *et al.* (2009), who delivered a solution for enterprises. Kemerlis *et al.* (2010) developed a lightweight system, called *iLeaks*, which dynamically traces information retrieval services. It consists of three main components, as shown in Figure 3.11: *Uaudits*, *Inspectors*, and *Trail Gateway*. *iLeaks* does not require kernel modification to intercept the system calls concerned because it utilises existing mechanisms in OS X, such as *DTrace* and data indexing services. These

services facilitate the tracing of computer program and process execution, along with its interaction with OS system calls. The authors describe the *iLeaks* working process as follows: once the *Uaudit* captures events that indicate a leak, it forwards them to the *Trail Gateway*, which, in turn, invokes the corresponding *Inspector* to verify that the candidate trail is indeed sensitive information that leaked into the network (Kemerlis *et al.*, 2010).

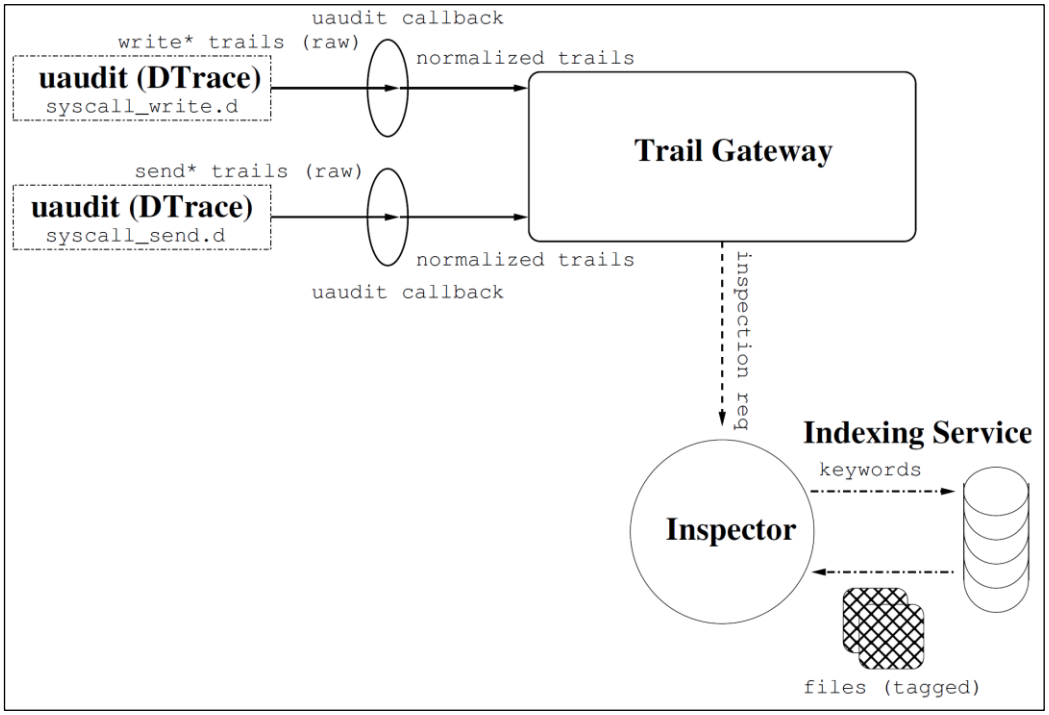

Figure 3.11: iLeak prototype implementation (Kemerlis *et al.*, 2010)

For the purpose of monitoring, however, the system relies on a predefined set of documents that are manually defined by the system user. In other words, the user must tag sensitive documents as being of interest in order to be monitored by the *Trail Gateway*.

In contrast, Ko *et al.* (2014) gave the computer user full control of the data flow during transmission, even when there is no predefined rule. Further, the designed software notifies the user what data are leaving the system and requests a decision to allow or block the transmission. By intercepting specific system calls, for instance, sys_*sendto:* for sending a file in the Linux OS, the system can detect suspicious activities that occur in the background. However, performing this kind of interception requires kernel hardcoding, as explained in previous studies reviewed in this chapter. Indeed, such low-level modification of the operating system code is not always applicable. For instance, Microsoft Windows and Apple OS X are both closed systems, which means access to the kernel code is not possible. Ko *et al.* (2014) do not provide sufficient information about the evaluation of this system, as it appears to be simply a theoretical model.

By trying to incorporate open-source data loss prevention systems, Koutsourelis and Katsikas (2014) developed a DLP system that takes advantage of two sets of free DLP software: MyDLP, (2014) and OpenDLP, (2014). The authors amalgamated a number of techniques in order to automate several operational processes that minimise user interaction with the system. Even with the many features that the system facilitates, it does not introduce any novel techniques or mechanisms that would help in detecting malicious data leakage. Instead, it merely employs existing methods. The study did not include any information regarding the performance evaluation of the developed system.

One of the most challenging data leak methods was investigated by Fujikawa, Mori and Terada, (2014). The authors studied the possibility of detecting the body

motion of an individual who is performing photo shooting or video recording (camera holding) to leak sensitive information. The system requires a motion detection sensor, which is attached to the computer screen. First, the sensor obtains three-dimensional information of the user's motion. Then, the system analyses the captured information to detect any camera holding. Finally, based on the detection result, if the system suspected a potentially malicious activity, the computer is locked and an alarm is triggered.

The above evaluation demonstrates that this technique is able to identify attempts at computer screen shooting. It can also distinguish between a camera being held by one or both hands. However, the system requires certain conditions to perform well, such as the subject being in a stable position (sitting at a desk) during photo/video shooting. This operation can be disrupted when an insider takes a screenshot from a different angle, such as standing outside the scope of the motion sensor, which significantly reduces the chance of detection. However, digital cameras, and even smartphones, commonly feature deep zooming technology that enables the shooter to take a high-resolution photo or video at various distances from the object.

3.3.5 Guilt Identification

Various studies have examined the detection of the leakage of sensitive data by a distributor's agents. Some of these studies have further investigated the possibility of identifying the agent that probably leaked the data (Papadimitriou and Garcia-Molina, 2011; Kale and Kulkarni, 2012; Jadhav, 2012; Chavan and Desai, 2013). It is important to mention that all these studies assume that the data

owner has full control over the data creation and distribution prior to their being given to the agents. One of the most cited studies in this field was conducted by Papadimitriou and Garcia-Molina (2011), who proposed an allocation strategy to increase the possibility of detecting leaked data. Furthermore, they introduced a model for assessing the 'guilt' of agents. The authors claimed that the proposed method does not rely on alterations to the released data, such as watermarking. This method considered two types of agent request, as presented in Figure 3.12: explicit and sample requests. An explicit request holds particular settings and all the items in the dataset that adhere to those settings must be returned. A sample request determines the number of objects to be randomly chosen from the entire dataset.

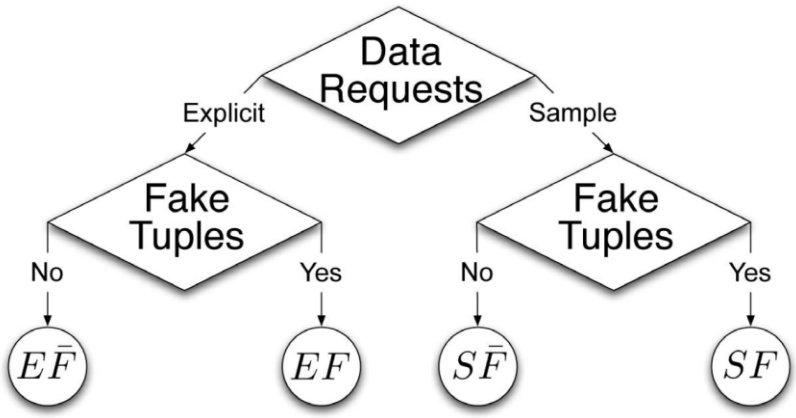

Figure 3.12: Leakage problem instances (Papadimitriou and Garcia-Molina, 2011)

Moreover, to increase the chance of identifying the agent responsible for a leak, Papadimitriou and Garcia-Molina considered the feasibility of inserting fake objects into requested data. The idea was to add a unique object to the data prior to handing them to the agent. However, adding fake objects is not always possible;

for instance, in the case of medical records, manipulating the data or injecting invalid information could lead to enormous risk. Therefore, the method is based on the nature of the data and how the fake elements generated modify the integrity of the original data. Examination of the feasibility of the system found that the proposed algorithms performed better in identifying the source of the data leakage compared with simple data allocation algorithms. Papadimitriou conducted several experiments, achieving scores with an average confidence level of 95%.

Subsequent implementations of the guilt model by Kale and Kulkarni (2012), Jadhav (2012), Chavan and Desai (2013), and Kumar *et al.* (2014) resulted in the development of several prototype models. All those models use the concept of inserting unique fake objects or digital watermarks into data prior to distribution. In general, the data creator (in this case the distributor) is responsible for generating and embedding the fake object, whereas, in many cases, the data can be created by an insider who leaks the sensitive data themselves. However, Jadhav (2012) admitted that the process of creating fake but real-looking objects is a complicated task and beyond the scope of that author's study.

3.3.6 Proactive Digital Forensics

The last part of this literature review discusses studies that provide some proactive functionality for digital forensics. The studies selected touch on the problem of data leakage from a forensic perspective, and the problem of identifying users who manipulate the use of sensitive files. For example, Cohen *et al.* (2011) developed a live, remotely accessed forensic solution that uses the

web interface for administering hosts and instantly accessing running computers. It is designed to be a centralised controlling framework and to support multiple operating systems, including Windows, OSX and Linux. Rafique and Khan (2013) argued that requiring a client-server environment makes the system less applicable for standalone machines and represents a limitation in Cohen *et al.*'s framework. Despite Rafique and Khan's argument, the proposed system facilitates a rich set of forensic analysis features, such as attributes related to an object (e.g., file memory location, accessed timestamp, AFF4 stream).

Cohen *et al.* (2011) also investigated a leakage of intellectual property scenario in which restricted information had been leaked. The aim of the investigation was to identify people who had access to that information. The task was performed by searching for specific keywords across hosts that run the system's agent, as the technique could determine a list of machines that had those keywords. Further investigation could then, as stated by the authors, successfully isolate users of interest. However, the scenario only leads to document allocation, without providing intelligence information that would inextricably link the use of the information to the individual who used and accessed it.

Similarly, Magklaras *et al.* (2011) developed an audit engine for actively logging user actions in a relational database management system (RDBMS). The proposed system could be used by post-case forensic examiners to aid an incident investigation. It stores actions, including files accessed (e.g., name, type, and location), timestamps, process execution, network endpoint and hardware device, in addition to other related information. Furthermore, the engine employs

a linguistic analysis of users' correspondence as a monitoring technique, thus proactively detecting potential insider threat risks in the organisation. It also facilitates the use of Structured Query Language (SQL), which enables instance selection and completion. This function allows investigators to enumerate databases and execute different types of enquiry. The system was tested on a variety of simulated insider misuse scenarios. Although the evaluation results are promising in terms of logging different types of user actions along with useful information, it still does not correlate those actions to the identity of the individual who performed them.

The final paper in this review is by Shields *et al*. (2011), who proposed a system, called *PROOFS*, that proactively and continuously collects evidence by creating and storing file signatures that are deleted, edited, or copied within the computers in the local network. The system uses a centralised database to store the signatures of the objects generated, which provides significant information, such as user ID, object timestamp, and the type of event. For instance, in events such as file creation or deletion, the signature contains the fingerprint (the bit-vector fingerprint of the document), user identifier, file name, file path, a timestamp for the event, and a machine identifier. This is especially helpful when conducting a forensic activity. The fingerprints generated are equal to around 1% of the storage space of the original file, which is a significant reduction in the size of the object. The system supports text files, such as Microsoft Word documents (.docx) and portable document formats (PDFs).

Overall, when the documents were shrunk by 60% of their original text size, the system was able to achieve an average recall rate of 95%, with a precision rate of 85%, of relevant documents based on the fingerprints generated, as illustrated in Figure 3.13. The experiment used the Enron email dataset, and a cosine similarity measure to measure the match between the queried fingerprints and the inspected documents. For its deployment, the system requires patching the system kernel in order to intercept system calls. Unfortunately, this low-level kernel hardcoding is limited to open-source operating systems, as explained earlier.

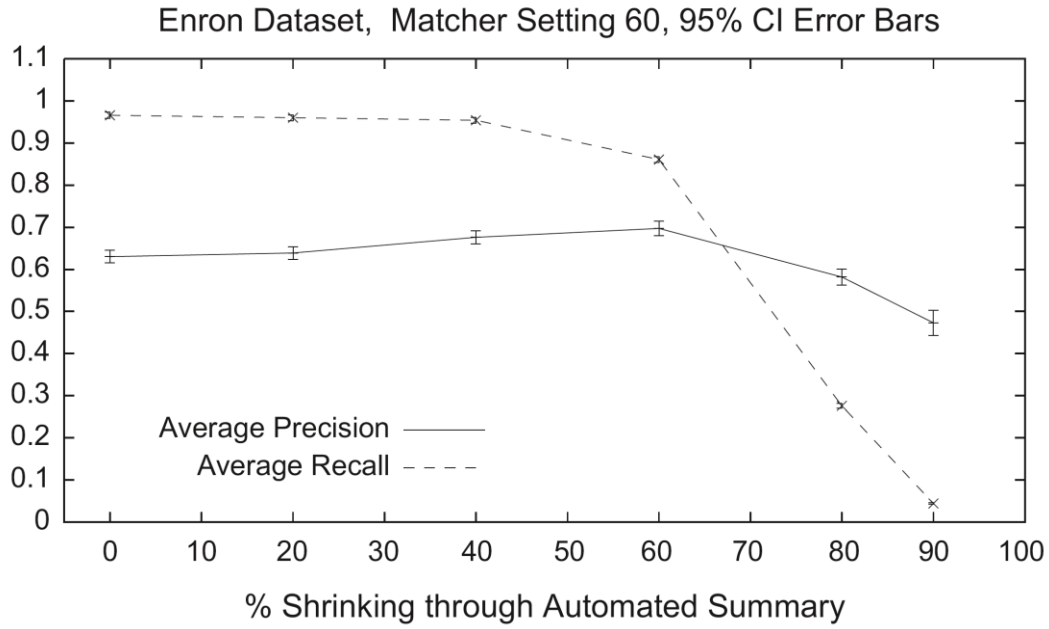

Figure 3.13: Enron automatic summarisation, matcher 60 (Shields, Frieder and Maloof, 2011)

3.4 Discussion

Academics have contributed to the field of DLD/P by proposing numerous methods and techniques to combat and reduce existing issues. A number of studies have investigated document classification to detect the existence of confidential information by clustering documents based on their similarities. Among those studies, fingerprint-based methods that use a cycling hashing approach attained a high performance compared with other classification algorithms, such as SVM, as examined by Katz *et al.* (2014). This is not surprising since fingerprinting is applied widely in different applications, such as plagiarism detection. However, fingerprint accuracy can be affected by text manipulation techniques when the confidential part of a text is rephrased and changed, which results in different versions in the training set. Other attempts have been made by researchers to detect data leaks via email by employing text and document classification, as examined in Liu *et al.* (2014) and Manmadhan *et al*. (2014), whereas both Kalyan and Chandrasekaran (2007) and Liu *et al*. (2014) proposed generic variables to detect the probability of data leakage. These methods were expected to generate a high rate of false positives, although the authors did not discuss this issue.

Research employing data encryption or encoding techniques to preserve confidential data has focused upon preventing outsiders from reaching them, as investigated by Sobh (2013) and Prakash and Singaravel (2015), although insider threats have been considered the utmost security issue by many recent research studies (Shabtai, Asaf, Yuval Elovici 2012; Collins *et al.,* 2013; Huth *et al*., 2013).

Encryption and access control significantly help prevent data from being leaked by untrustworthy subjects. However, the focus of such research is on protecting the confidentiality aspect of the data, rather than focusing on tracing the source of the leakage. Mainly when the case involves insiders who supposedly have legitimate access to highly sensitive information by providing their credentials, encryption and encoding mechanisms fail to control the intent of the subject. Consequently, a user could gain full control over the data and a malicious user could even use encryption schemes to bypass intrusion detection systems.

Interestingly, studies such as those by Lee *et al.* (2013) and Wu *et al.* (2013) have tried to use biometric information as a factor in detecting the malicious activities of an insider. It is clear, however, that the experimental setup is rather controversial, and there is neither an indistinct, strong correlation between their hypotheses nor the actual implementation of the data field of DLD. In addition, linking the subject with leak incidents is not a simple task to achieve, even with methods such as the guilt identification model, as this model is designed precisely for a case in which a data distributor has given sensitive information to a group of supposedly trusted agents (Papadimitriou and Garcia-Molina 2011). Therefore, the distributor should have full control over the distribution process, which is not always possible; for instance, insiders could have a legitimate right to access confidential data without the need to request it. Similarly, studies using proactive forensic solutions that take into account the issue of DLD can potentially offer some functionalities for linking leakage incidents with the suspected user.

Regardless of whether the object is in the form of an image, document or email message, it appears from the studies as mentioned above that numerous investigations have been conducted on the detection and prevention of data leakage. However, no attempt has been made to investigate the possibility of linking an incident to a suspicious subject in a biometric manner. Digital forensic investigators could benefit from linking the identity of a digital object to an individual, as opposed to simply using an electronic record or a log that indicates a user interacted with the object in question (evidence). Indeed, this is a challenging task, as it is currently difficult for digital forensic professionals and investigators to prove beyond a reasonable doubt in a court of law that a specific human being used the particular identity of a digital subject at a particular time (Shavers, 2013; Brown, 2015; Vincze, 2016). Biometric techniques could provide such a link to correlate the user interaction with the used object, thus giving rise to relevant information that would help investigators to answer the question: “Who did the crime?”

The next chapter conducts an investigation into a biometric-based null cipher technique using images, in which the object that has been interacted with is linked to the individual who accessed it.

3.5 Conclusion

As presented in this chapter, several methods and systems from different perspectives have been proposed for solving the problem of data leakage. Overall, it is challenging to compare the studies reviewed with each other adequately in terms of evaluation performance due to the variation between the studies, as the

authors used different metrics and datasets to present the results of their experiments. The outcomes of this review also exposed an absence of significant literature on the specific aspects of the topic of investigation of this research.

The existing literature has largely failed to overcome the key problem of being able to associate a specific person with electronic evidence and, as a result, stolen credentials and the Trojan defence are two commonly cited defences. There is, therefore, an urgent requirement to develop a technique that can inextricably link the use of information (e.g., documents and emails) to the individuals who access it, rather than intermediate controls. The next chapter introduces a proposed approach, which leverages biometrics technology along with steganographic techniques to provide this link.

4 System Design and Use Cases

4.1 Introduction

This chapter describes how the system components work to correlate a leakage incident with the person who is responsible along with potential attack vectors. As shown in Figure 4.1, Actors such as user, outsider and digital forensics investigators are involved within a given scenario within which use cases is descript to explain how the incident and system reacts systematically.

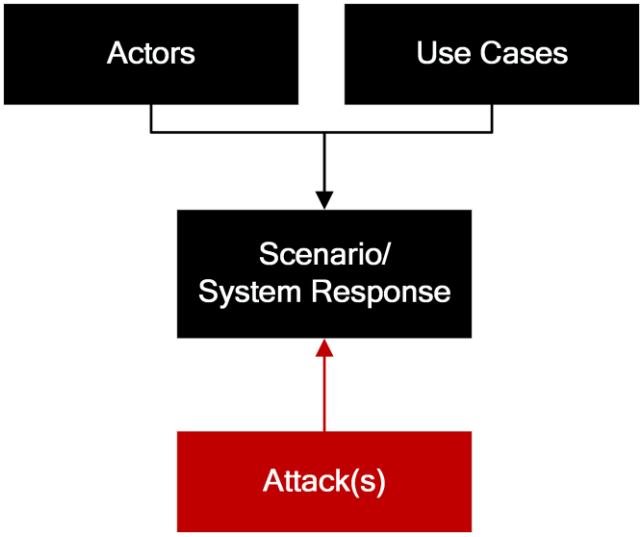

Figure 4.1: Use Case Scenarios Structure and Logic

The elements of Figure 4.1 above work together to describe the system functionality both under normal operation and under attack conditions. These elements are linked to the use cases to describe how the system operates in such different cases.

4.2 Description of Taxonomy

To derive accurate scenarios and use cases, a system engineering approach has been used whereby actors have been derived and will be used throughout the use cases. Table 4.1 illustrates the main actors in the system.

Table 4.1: Common Person-Class Actors

ID	Description
P1	An Insider/user who works at the entity/company who owns that classified/secret data
P2	An outsider who might receive a leaked data (e.g. document/image)
P3	An investigator who can use the system to extract correlated suspect information

In Table 4.2 are the three categories of data collected as Cyber-Trust deals with different categories of data.

Table 4.2: Common Data-Class Actors

ID	Name	Description
D1	File data	Reflects data related to digital objects (e.g. documents/images) such as file content data, fingerprint and hash digest.
D2	Biometrics data	Reflects computer data that is created during a biometric process. This includes samples, models, fingerprints, similarity scores and all verification or identification data

D3	Forensic data	Reflects the collection, processing and storage of information that may contain evidentiary material

Because it is necessary to link the elements to corresponding use cases, Table 4.3 places current envisioned components into actor roles:

Table 4.3: Common Asset-Class Actors

ID	Name	Description
A1	Biometrics engine	Captures and extracts the user's biometric samples and stores it in a database on the user's computer
A2	Imprinting engine	Retrieves the object metadata and a recent biometric sample(s) from the biometric engine and perform the imprinting process
A3	Biometrics centralised database	Stores users' biometrics (e.g. extracted facial feature vector)
A4	Interactions/logs centralised database	Stores the generated imprints and individuals' along with user-object interaction logs
A5	Generic electronic device	This includes computers, smartphone, USB thumb drive, web-camera…etc.

Table 4.4: Common Attack-Class Actors

ID	Name	Description
T1	File-type conversion attacks	Reflects attacks in which a file type/extension is changed into another (e.g. docx to pdf or jpg to png)
T2	Document Formatting change attacks	Reflects attacks in which a format/style of a document's text is changed (e.g. change in font size, font type, colour…etc.)
T3	Content manipulation attacks	Reflects attacks in which file content is changed/modified (e.g. remove part(s) of the file, cropping an image…etc.)

4.3 Main Scenarios

To effectively develop the use cases, this chapter defines the main scenario, which then is detailed with the possible attack vector(s). The specific system functionality will then develop and shape the associated use cases, according to the main scenario, and describes how their system element will respond to the attack vectors described below.

4.3.1 Leaking Classified Documents/Images

This real scenario reflects a case in which an insider employee [P1] leaked a classified document [D1] to a reporter [P2] in the United States. Hale [P1] served as an enlisted airman in the United States Air Force after receiving intelligence and language training, Hale has been assigned to work in the National Security Agency (NSA) and deployed to Afghanistan as an intelligence analyst. According

to Hal's active duty service and function for the NSA, Hale's period in NGA, Hale held a leading Secret/Sensitive compartmented information security clearance and has been entrusted with access to classified national personal information [D1] Hale published six classified documents [D1] to a reporter [P2]. Each one of those six records [D1] was afterwards published by the reporter publicly (Stuev, 2019). These leaked documents [D1] could be imprinted [UC3] with Hale biometric information [D2] using use case UC1 if the proposed system was deployed and operated in place. Hale owned a thumb drive [A5], which was analysed by a digital forensic investigator [P3], and it has been found out that it comprised a classified document [D1] that Hale had published in February 2014 and had tried to delete [D3] in the thumb drive [A5] (Stuev, 2019). Use case [UC6], could be used to retrieve the imprinted suspect biometric information [D2] in which correlated and more information can be established and extracted.

4.3.2 Leaking and Modifying Classified Documents/Images

The following scenario includes leaking and modified a classified document [D1]. As Reality Winner [P1], a 25-year-old Air Force veteran and federal contractor sent journalists a top-secret NSA intelligence report [D1], which showed that the NSA had collected intelligence suggesting that Russian military intelligence had tried to gain access to states' electronic voting systems in 2016. The Intercept published [P2] a partially-redacted version of the report [T1, T2, T3] on June 5, 2017 (Pressfreedomtracker.us, 2019).

These two scenarios can be correlated with the isdentifed use cases and actors as shown in Figure 4.2.

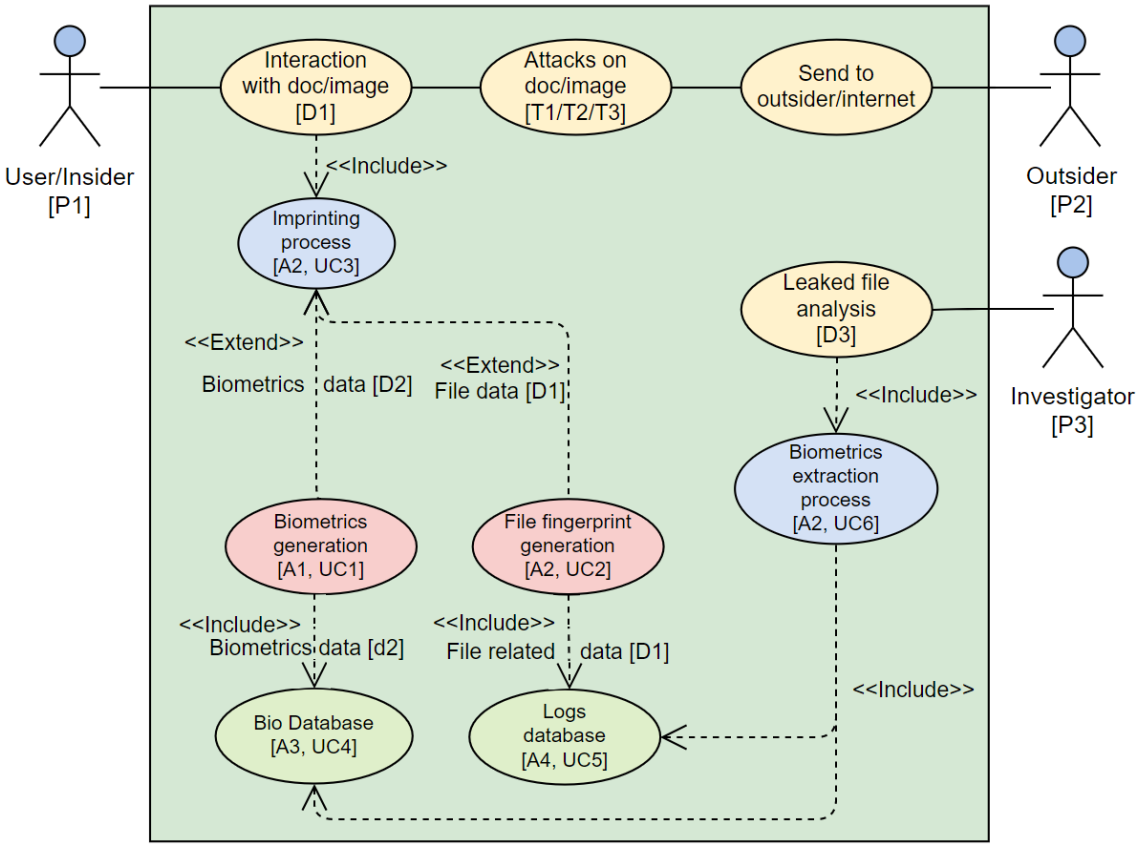

Figure 4.2: General file leak scenario including Use Cases and relevant Actors

4.4 Use Cases

The use cases presented in this section composed of the functional components of the system. The use case covers incidents, and the respective scenarios of each explained in the previous section. These use cases will be the stepping stone to produce in the system the architecture. In total, five use cases are identified in the table below. These use cases are explained in detail in section 4.4.2.

Table 4.5: Use cases description

UC ID	Name
UC1	Biometric feature vector generation
UC2	File fingerprint generation
UC3	Imprinting process
UC4	Storing biometrics in database
UC5	Storing file fingerprint in the database
UC6	Post-incident biometric extraction process

4.4.1 Use Cases-Actors Inter-Relationships

The main actors of the use cases are defined in section 4.2 of this chapter, and they are detailed as primary or secondary actors across the use cases in section 4.4.2 below. In addition, each use case details the inter-relations between the use cases to provide a meta-picture within each use case of the dependencies resulting between the components of the system.

	A1	A2	A3	A4	A5
UC1	X		X		X
UC2		X		X	
UC3	X	X		X	
UC4	X		X		
UC5		X		X	
UC6	X		X	X	

Figure 4.3: Use cases-actors inter-relationships

4.4.2 Use Case Detailed Specifications

ID: UC1

Name: Biometric feature vector generation use case

Description: User [P1] facial image(s) [D2] is captured by fitted camera [A5] and the biometric engine [A1] processes the captured image(s) to generate a biometric feature vector [D2].

Primary Actor: [A1] biometric engine

Supporting/Secondary actors: [A5] camera device and [P1] user

Stakeholders	Interest
[A1] biometric engine	Process and generates user's biometric vector
[A5] camera device	Captures user's frontal facial view

Frequency of use
Every time the user uses the device (e.g. computer)

Related use cases
[UC4] stores biometrics in database

Example
A user [P1] is set in front of a computer screen which is fitted with a camera device [A5] and started interacting with computer files [D1]. The camera [A5] capture the user facial image and is sent to the biometric engine [A1]. The biometric engine analyses the captured image and generates feature vector [D2], which then stores in the biometric database [A3].

Main Scenario

Step	Actor	Action description
1	[P1]	User starts using and interacting with computer files (e.g. document/image)
2	[A5]	A fitted camera device captures user facial image
3	[A1]	The biometric engine process the user's captured image and extracts the biometric features vector
4	[A3]	The generated feature vector is stored in the biometric database

ID: UC2

Name: File fingerprint generation use case

Description: When a user [P1] interacts with a computer file [D1], one or more fingerprints are generated by the imprinting engine [A2] to be used for establishing a correlation log.

Primary Actor: [A2] Imprinting engine

Supporting/Secondary actors: [D1] file data

Stakeholders	Interest
[A2] imprinting engine	Process and generates files' fingerprint(s)

Frequency of use
Every time the user interacts with a computer file (e.g. document/image)

Related use cases

[UC5] stores file's fingerprint data

Example
A user [P1] is interacting with computer files [D1]. The imprinting engine [A2] analyses the files and generates fingerprint/local sensitivity hashing digest [D2] which then stores in the fingerprint/logs database [A4].

Main Scenario

Step	Actor	Action description
1	[P1]	User starts using and interacting with computer files (e.g. document/image)
3	[A2]	The imprinting engine analysis and process files and generates a fingerprint
4	[A4]	The generated fingerprint is stored in the fingerprint/logs database

ID: UC3

Name: Imprinting process use case

Description: A computer user [P1] interacts with a computer file [D1], the imprinting engine uses the generated (one or more) fingerprints and biometric feature vector [D2] establishing a correlation log (i.e. imprint file).

Primary Actor: [A2] Imprinting engine

Supporting/Secondary actors: [A1] biometric engine

Stakeholders	Interest
[A2] imprinting engine	Process and generates imprint file

Frequency of use
Every time the user interacts with a computer file (e.g. document/image)

Related use cases
[UC1] Biometric feature vector generation use case
[UC2] File fingerprint generation use case

Example
A user [P1] is interacting with computer files [D1]. The imprinting engine [A2] generate an imprint file in which it uses one or more of the generated file fingerprints [D1] and user's biometric feature vector [D2].

Main Scenario

Step	Actor	Action description
1	[P1]	User starts using and interacting with computer files (e.g. document/image)
3	[A2]	generate an imprint file in which it uses one or more of the generated file fingerprints and user's biometric feature vector
4	[A4]	The generated imprint file is stored in the fingerprint/logs database

ID: UC4

Name: Storing biometrics in the database use case

Description: When the biometric engine [A1] generates a biometric feature vector [D2], the system stores it in the biometrics database [A3].

Primary Actor: [A3] biometrics database

Supporting/Secondary actors: [A1] biometric engine

Stakeholders	Interest
System	Stores the generated biometric data in the database

Frequency of use
Every time the user interacts with a computer file (e.g. document/image)

Related use cases
[UC1] Biometric feature vector generation use case

Example
A Biometric feature vector [D2] is generated by the biometric engine [A1]. The system stores the generated data in the biometrics database [A3].

Main Scenario

Step	Actor	Action description
1	[A1]	A user biometric feature vector is generated by the biometric engine.
2	[A3]	The generated biometric vector is stored in the biometrics database

ID: UC5

Name: Storing files fingerprints in the database use case

Description: When the imprinting engine [A2] generates a files fingerprints [D1], the system stores it in the fingerprints/logs database [A4].

Primary Actor: [A4] fingerprints/logs database

Supporting/Secondary actors: [A1] biometric engine

Stakeholders	Interest
System	Stores the generated file fingerprints in fingerprints/ logs database

Frequency of use
Every time the imprint engine generates file fingerprints

Related use cases
[UC2] **File fingerprint generation use** case

Example
File fingerprints [D1] are generated by the imprinting engine [A2]. The system stores the generated fingerprints in **the fingerprints/logs database [A4].**

Main Scenario

Step	Actor	Action description
1	[A2]	The imprinting engine analysis and process files and generates a fingerprint
2	[A4]	The generated fingerprint is stored in the fingerprint/logs database

ID: UC6

Name: Post-incident biometric extraction process

Description: After a document [D1] is leaked and obtained by a digital forensic investigator [P3] for analysis, the document is processed and the correlated

imprint file is located by the system by analysing the forensic data [D3] and the relevant biometric feature vector [D2] is extracted.

Primary Actor: System

Supporting/Secondary actors: [A3] biometrics database, [A4] logs database

Stakeholders	Interest
System	Analysis on the leaked document

Frequency of use
Every time a leaked document is analysed

Related use cases
[UC4] stores biometrics in database
[UC5] stores file's fingerprint data

Example
A classified document and an image file [D1] are leaked by an insider employee [P1] to a repository website (e.g. WikiLeaks) [P2], the document has been modified and its content is manipulated [T1, T2, T3] to remove any potential tracking/embedded data (e.g. watermarks/user-related data) that could lead to the source of the leakage. A digital forensic investigator obtained a copy of the leaked file [D1] and performed a forensic analysis to reveal who is responsible for the leak. The investigator began the analysis by generating a fingerprint of the file to be compared with the correlated imprint file by retrieving the logs from the database [A4] in which a correlation can be established and a biometric data [D2] can be located once there is a positive match.

Main Scenario

Step	Actor	Action description
1	[P3]	An investigator gets a copy of a leaked file
2	[A2]	File fingerprint is generated
3	[A4]	The generated fingerprint is compared with those stored in the fingerprint/logs database.
4	System	Once there is a match, the biometric information is reconstructed using the imprint file that is intact with the matched fingerprint/log.

4.4.3 Legal, Ethical and Privacy Dimensions

The impact on an individual´s privacy should always be assessed on a case-by-case basis. The system capabilities and functions should be only activated by specific incidents that are more likely to correlate to suspicious activity. For example, a user is accessing a classified document, and at the same time there is a USB memory stick plugged into the computer, in such case, the relevant engines (biometrics and imprinting) should be activated by the system.

Data subject´s rights are fully communicated to the users. Users should be able to understand the implications of the system, detailed data protection and privacy statement should be easily accessible, including information about which types of data is collected, for which time-period, who can access it. The storage and transmission of data collected must be safeguarded. For example, by using suitable integrity, encryption algorithms and safe network transmission protocol.

It must be ensured, that only persons with the right authorisation can access information generated by the system, which might include personal data,

individual's biometrics and performed activities (e.g. user interactions with computer files). The participation of external entities/parties to the system (e.g. digital forensic investigator) and their access rights must be elaborated accordingly, to minimise the possibility of false positives or the likelihood of access to a user´s data without the right authorisation.

4.5 Conclusion

This chapter identifies system actors, use cases and a relevant scenario to define, investigate, and evaluate the system would react in normal operation. It is worth noting that the design of the use cases aims to provide scenarios to conceptually stress-test the proposed system, rather than act as an exhaustive validation set. Both use cases and actors are applicable across all the scenarios and described as part of the methodology that also introduces the system design in the following chapter. Chapter 5, explains the proposed system and its components and capabilities in more details.

5 Proactive Biometric Imprinting of Digital Objects

This chapter introduces a proactive digital forensics biometric-based approach to the attribution of misuse via information leakage, using biometrics and steganography. Two main methods are discussed: null cipher and grille cipher imprinting techniques. The chapter concludes with the main research questions investigated experimentally in subsequent chapters.

5.1 Introduction

From the exploration of the existing insider misuse tracing techniques conducted in the previous chapter, a limited number of studies have tried to leverage soft biometric signals in detecting malicious insiders' activities (Almehmadi and El-Khatib, 2014; Lee *et al.*, 2014; Hashem *et al.*, 2015). Those studies proposed systems that employ the use of human bio-signals, such as electroencephalography and electrocardiogram, to detect insiders' malicious activities. For detection, they measure the difference in bio-signal deviations between normal and malicious activity phases. Although both systems deployed their approaches in real-life scenarios and achieved high detection accuracy, the experimental setups relied on users wearing a headset that continuously monitors bio-signals and a finger sensor to capture them. It is both unrealistic and un-user-friendly to wear these sensors continuously in real life. None of the current literature has specifically sought to embed individuals' physiological and behavioural biometric signals transparently into digital objects, such as images, other than one study that mentioned a grille cipher (Alruban *et al.*, 2016).

5.2 Steganography and Transparent Biometrics

Several studies successfully leveraged transparent biometric techniques for the purpose of user authentication, verification and profiling (Clarke, Karatzouni and Furnell, 2008; Saevanee *et al.*, 2015; Al Abdulwahid *et al.*, 2016; Al-Bayati *et al.*, 2018; Abed *et al.*, 2019). Such techniques can also be utilised to acquire biometric signals from individuals transparently as they interact naturally with a computer system and then embed the captured signals within digital objects, such as documents, PDFs, emails and photographs. In this manner, the last individual to access a digital object will have their biometric information bound to it. Subsequent misuse of the information, through disclosure, for example, would enable a legitimate organisation to process the digital object, recover the biometric identifiers and identify the last employee who accessed it. The transparent acquisition of biometric information is key to the solution, as an employee will not knowingly provide samples in an intrusive manner (i.e., the system will never ask the employee to provide a fingerprint sample or look at a camera) (Clarke, 2011). Rather, the approach seeks to employ a range of biometric modalities and continuously acquire biometric samples in a frictionless manner while the user is interacting with the system; for example, facial recognition using a web camera while the user is responding to an email or surfing the Internet. A multimodal and transparent/frictionless approach significantly reduces the opportunity for forgery and circumvention. The frictionless nature of the biometric capture also means that no action is required on the part of the user—thereby removing any additional workload or effort.

This research proposes an approach that is designed to operate in one of two modes: centralised or decentralised. Whilst it is envisaged an organisation is likely to select one of the two modes, the approach is capable of simultaneously operating in both—with the organisation being able to select which types of digital object use for which approach. For example, a document classification scheme could be used, with lower classification approaches utilising a decentralised mode and more highly classified documents using a centralised approach. The two modes offer a different level of repudiation but also introduce additional overheads in terms of storage and processing of data.

Conceptually, the centralised approach seeks to provide a mapping between a digital object and biometric identifiers, storing the mapping information alongside document identifiers in a centralised storage repository. When objects are recovered or analysed, the information stored in the repository is used to recover the biometric information which is subsequently used to identify the user. The key advantage of this approach is that the underlying digital object is not modified in any form. Whilst the nature of typical watermarking or steganography processes is not to modify the digital object in a manner that is noticeable, it does still modify the document and there may be situations in which this modification is not desirable. It also disassociates any biometric information from the digital object itself, thereby minimising any attacks against the biometric data. This approach allows larger volumes of information to be embedded, making it more suitable for digital objects that are smaller or when greater levels of information need to be embedded (i.e., multimodal biometric samples). It does, however, introduce the

need for a centralised repository, which will grow as users interact with objects and will thus require configuration and management.

The decentralised approach seeks to overcome the need for centralised storage by embedding all the necessary information within the digital object itself. This will result in the modification of the object but only in a manner that is not visually noticeable. The organisation need only process the recovered object to obtain the biometric identifier and identify the user. The amount of biometric information that can be embedded is directly linked to the digital object, introducing a trade-off between embedding a uni biometric or multimodal biometric identifier.

The key to this proposed solution is its ability to recover biometric identifiers successfully, even under significant modification attack. Rather than requiring the complete digital object, it is possible to recover the necessary information with only snippets of the original document.

The following sections provide a detailed breakdown of the decentralised and centralised approaches using null-ciphered and grille-ciphered steganographic techniques, respectively.

5.3 Null-Ciphered Imprinting Using Images

The decentralised approach is a secure and portable method that can link individuals with an object with which they have interacted (e.g., digital images) without the need for a centralised database. The correlation between the computer user and the digital object is established by leveraging biometric signals in a transparent fashion, along with a steganographic technique for embedding it

into the object. Figure 5.1 illustrates the decentralised approach elements and the main process flow.

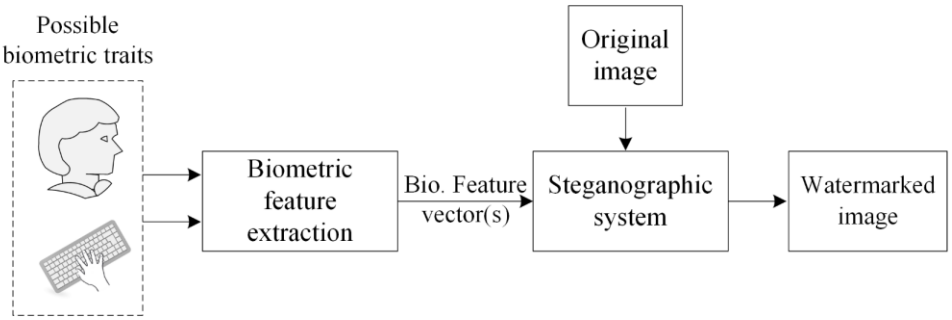

Figure 5.1: Decentralised process

The embedding procedure, as illustrated in Figure 5.1, follows the following steps:

1. Capturing and extracting the individual's biometric information.
2. Transferring the biometric signal and image into a byte stream and adding the required padding, flag and timestamp.
3. Encrypting the payload.
4. Selection and embedding of the encrypted data into the file.

A detailed description of the embedding and extraction process is provided in Chapter 6.

Figure 5.2 schematically represents an image file that shows image data *c* and flag data contained in two portions, *a*, *b,* which together form the flag and data to be embedded. On creating or handling file *c*, biometric information corresponding to the creator or handler of file *c* is processed by the computer in which file *c* is

created or handled to generate a digital biometric feature vector file. Data portions corresponding to parts of the digital biometric feature vector file are then distributed amongst and embedded within other data within flag portion *a* according to a key, and key data corresponding to the key is stored or embedded within header portion *b*. The key corresponds to locations within flag portion *a* where the data portions are embedded. In order to associate file *c* with the creator or handler of the file, the flag portions *a*, *b* are processed to reconstruct the digital biometric feature vector file. The reconstructed digital biometric feature vector file can then be compared with each set of examples in digital biometric feature vector files held in a database in order to identify the creator or handler of file *c*. When the digital biometric feature vector file of the creator or handler of file *c* is processed into flag portion *a*, it may at the same time be added to the database so that the database is dynamically updated.

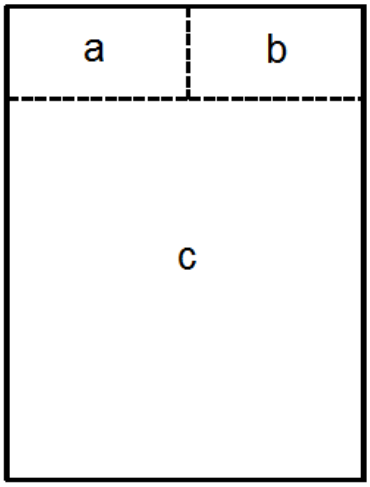

Figure 5.2: Null-ciphered image

The watermarking/steganographic process was specifically designed to counter file modification attacks, resulting in robust and reliable retrieval of biometric information.

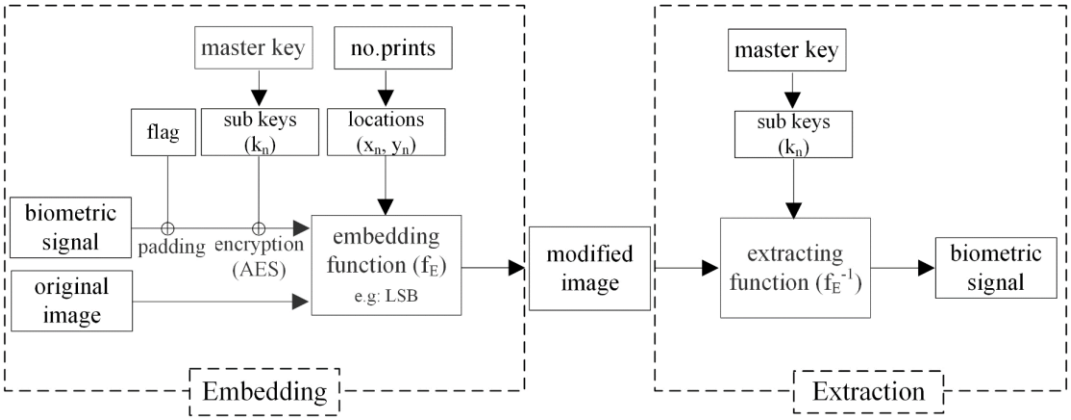

Figure 5.3: Embedding and extraction procedures used by the steganographic system

To maintain the privacy of the biometric data, all data are encrypted prior to the watermarking process. Only a legitimate organisation will have the ability to process, identify, extract and biometrically identify an employee from an electronic file.

The solution mitigates against the need to collect, decrypt and store large volumes of network, server and application logs and provides a lightweight approach that embeds the necessary information within the file itself. Apart from an organisation needing to store a copy of an individual's biometric data securely against which future data can be compared, no other information or continuous centralised storage is necessary.

5.4 Grille-Ciphered Imprinting Using Images and Text Files

The centralised approach consists of two engines: a biometric and an imprinting cipher engine. The biometrics engine transparently captures and extracts the user's biometric samples and stores them in a temporary database on the user's computer. The imprinting engine retrieves the object metadata and a recent biometric sample(s) from the biometric engine to be used in the imprinting process. Finally, the imprints generated are stored in a centralised database for later analysis when required. Figure 5.4 illustrates the framework architecture of the centralised approach.

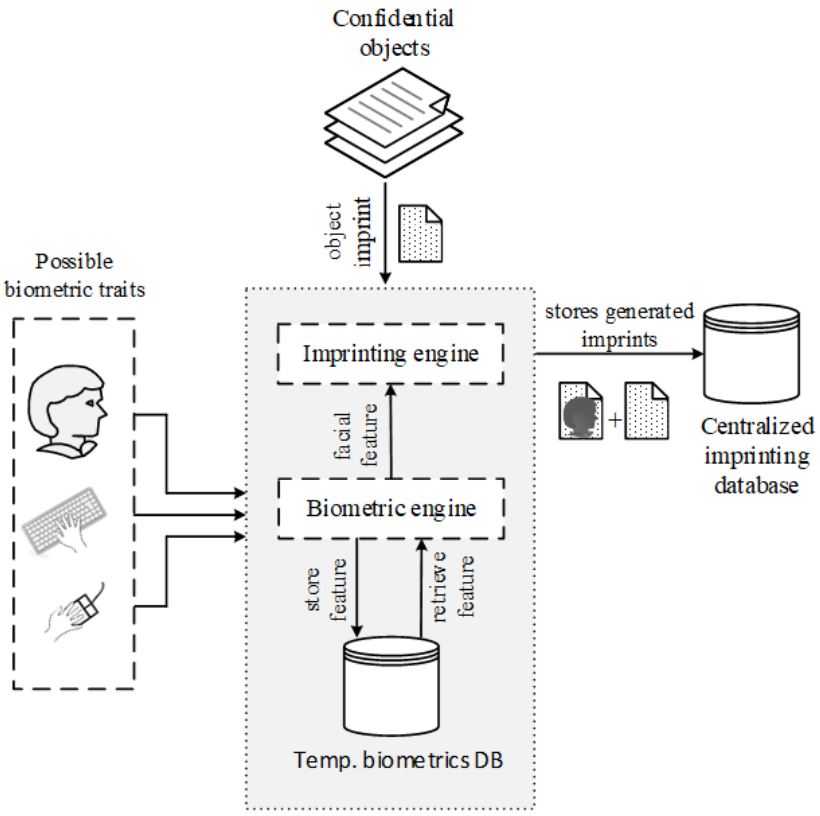

Figure 5.4: Centralised process

The most important process in the centralised approach is mapping the biometrics signal(s) with the object. A detailed description of the mapping process is given in chapter 7 and 8.

Upon the detection of data leakage, the object (whether it be posted on a public website or captured by the network) can be analysed for its biometric imprint. The sample is extracted and then processed by a biometric system in order to determine the last user to interact with the object, as presented in Figure 5.5.

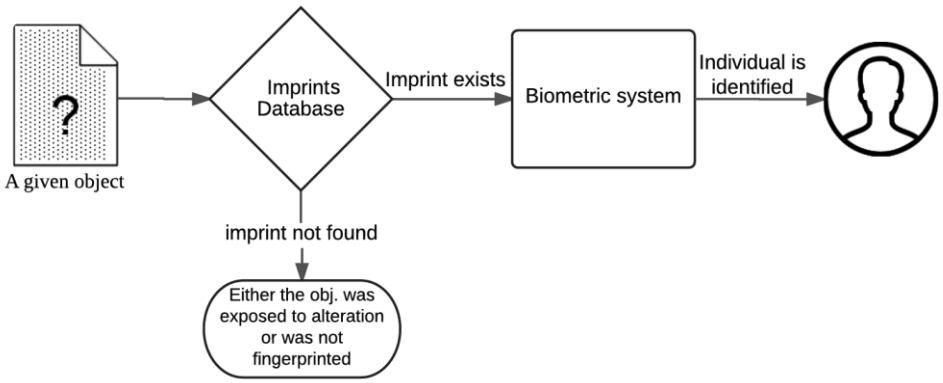

Figure 5.5: Process of identifying an individual

5.4.1 Digital Object Imprinting

The imprinting process technique takes a digital object file, such as an image or text file, and processes it with a biometric feature vector file that corresponds to an individual to produce a series of digital imprint files that are kept in centralised storage. A given digital imprint file represents locations within the original electronic file, each of which corresponds to a respective portion of the biometric feature vector file. A given electronic file, which may be the original file or a modified version of it, is processed with each of the digital imprint files to generate

the respective reconstructed biometric feature vector file. The reconstructed biometric feature vector files are compared with the biometric feature vector files in storage to associate the individual with the given electronic file. The method allows a particular digital object file to be linked to an individual with greater certainty than is possible with the use of fake objects, watermarks or digital signatures, even if the file was created following substantial modification of the original electronic file.

5.4.2 Correlation Chaining

The proposed approach can also serve an application other than mitigating malicious insiders, such as by embedding a person's biometric trait signal (e.g., facial or iris biometric information) into an image being captured by, for instance, a smartphone or computer webcam. This can also be extended to handle a chaining scenario, in which more than one individual's biometric information can be linked and stored once they have interacted with a traced image.

Figure 5.6 shows a schematic of an example application of how a null-ciphered approach can be applied to a smartphone scenario. The first person (A), has a smartphone (not shown), which stores the biometric information of the first person and updates it over time, such as daily, weekly or monthly. The biometric information includes one or more of the fingerprint characteristics captured during logging-in, voice characteristics captured during phone calls and/or the operation of the speech recognition functionality of the smartphone and facial characteristics (possibly including iris characteristics) captured with a rear-facing built-in digital camera in the smartphone. A biometric feature vector file (FV) for

the person A is generated and updated over time using the biometric information of the first person captured by the first person's smartphone and the biometric feature vector file is stored on the first person's smartphone.

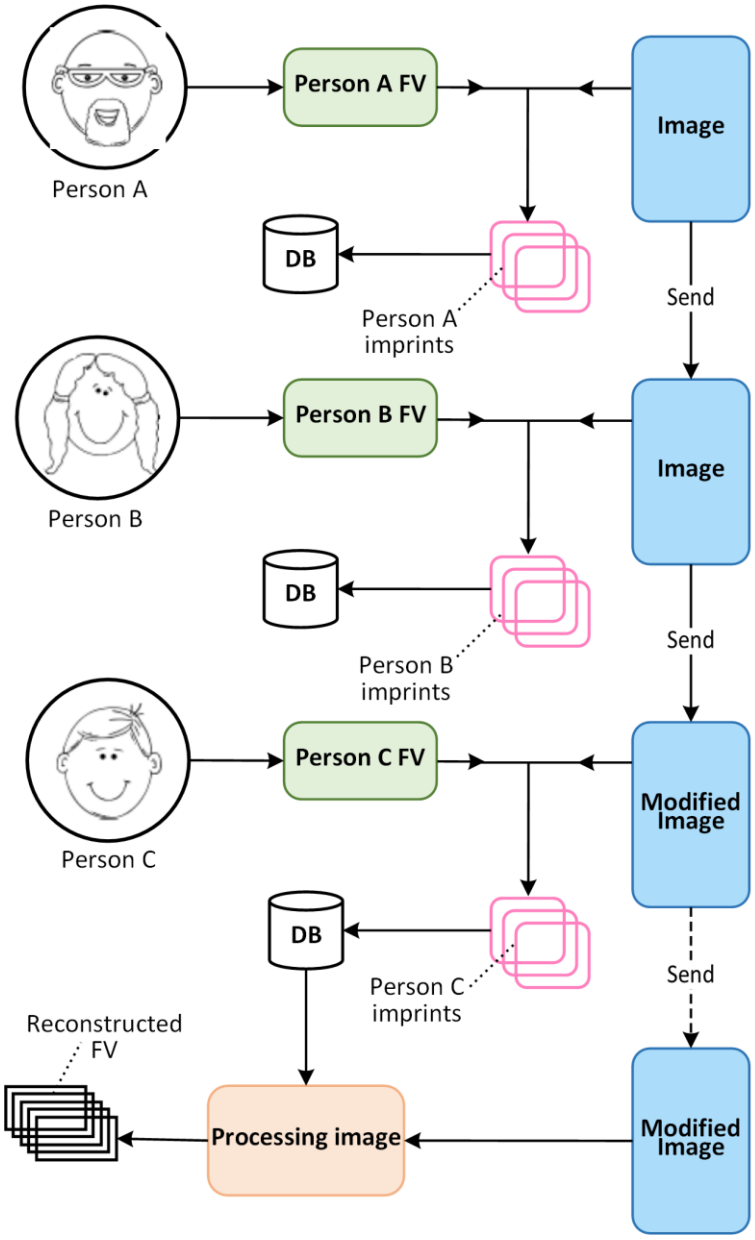

Figure 5.6: Example of linking multiple persons' biometric information with an image

Person A creates an image file using a built-in forward-facing digital camera in the smartphone. Immediately after the image file is created, a biometric feature vector file is processed, together with the image file, as described above, to generate a set of digital imprint files which is stored in a central database. The digital imprint files are transmitted over a wireless network to the central database, which is maintained at a remote location. Person A then transmits the image file by email or MMS to the smartphone of a second person (B). The second person's smartphone (not shown) captures biometric information from the second person and maintains a regularly updated biometric feature vector file on the second person's smartphone using that biometric data. Person B opens the file but does not modify the image data of the image file. On opening the image file, the second person's smartphone operates to process that person's biometric feature vector file with the image file to produce a second set of digital imprint files, which is added to the central database. The second person then uses her smartphone to send the image file to the smartphone of a third person (C).

The third person's smartphone (not shown) operates to maintain a biometric feature vector file using the biometric information of the third person, captured by the third person's smartphone. The third person opens the file and modifies the image data of the image file to produce a modified image file. The modified image file may, for example, be produced by cropping the image data, pixelating one or more parts of it, combining it with other image data, or by a combination of two or more such operations. The third person's smartphone operates to process the biometric feature vector file with the modified image file to generate a third set of digital imprint files, which is added to the central database. The modified image

file is then subsequently transmitted by the third person to further recipients in turn whom each handle the modified image file (e.g., opening, closing, saving or transmitting the file), either with or without further modifying the image data of the file.

The above steps result in an image file comprising image data related to those of file. The identities of individuals, such as A, B and C, who have handled and/or modified earlier versions of image file are identified by processing the image file with each of the digital imprint files held in the central database to generate a set of reconstructed biometric feature vector files from which the individuals (A, B, and C) can be identified.

Starting from a given electronic file, the identity of someone who has handled the file (either with or without modifying the file's image data) can be recovered by processing the electronic file with digital imprint files held in the central database to generate a set of reconstructed biometric feature vector files.

The reconstructed biometric feature vector files may be processed with each of a set of example or candidate biometric feature vector files held in a separate database in order to identify individuals. That database can be compiled by adding digital biometric feature vector files to the database by transmitting them over the wireless network after the users' smartphones generate them.

5.5 Discussion

The proposed system aims to facilitate a proactive approach to tracing digital object leakages by linking individuals' biometric signals to the objects with which

they interact. On the other hand, the current approach to detecting insider misuse—as explored in chapter 3—involves a layering of security countermeasures that include comprehensive logging of servers (including authentication requests), which suffer when encryption is in place, proxy-based network decryption and the storage of network traffic, which are required to identify misuse, possibly over prolonged periods of time (Pilli, Joshi and Niyogi, 2010; Birk and Wegener, 2011; Khan *et al.*, 2016). However, there are a number of approaches that utilise steganography and watermarking techniques to embed specific data that could point to the action generator (Chaabane, Charfeddine and Ben Amar, 2013; Macq, Alface and Montanola, 2015; Bouslimi and Coatrieux, 2016; Alruban *et al*, 2017). While conventional watermarking or steganographic processes do not modify the digital object in a noticeable manner, they nonetheless alter the document. There may be situations in which this modification is not desirable, for instance, when preserving the integrity of the object is crucial.

In contrast, the approach proposed in this study seeks to provide a mapping technique between the digital object and biometric identifiers, storing the mapped information alongside object identifiers in a standalone and centralised storage repository. When the mapped (imprinted) objects are recovered or analysed, the information stored in the repository is used to recover the biometric information that is subsequently used to identify the user. The two key advantages of this approach are that it leverages transparent biometrics to establish the desired correlation between user and object and the underlying digital object is not modified in any way, in contrast with the aforementioned watermarking and logs-

based studies. In addition, no explicit biometric information is stored, as only the correlation that points to locations within the imprinted object is preserved.

As stated above, the proposed approach provides two main methods for attributing insider misuse by creating a correlation between individuals and the digital objects with which they have interacted. The novelty of these approaches can be summarised in the following points:

- The system serves the frictionless capture of biometric identifiers, resulting in a usable yet secure approach to associating users with the digital objects with which they interact.
- The multimodal design enables broad participation and mitigates against forgery and circumvention.
- The centralised and decentralised approaches provide flexibility and differing levels of repudiation capability.
- The system offers the capability to recover biometric identifiers under significant levels of modification attack.

While the introduction of such a system has the foundations for providing such a link, several concerns and issues need to be considered and addressed before the ubiquitous adoption and effective operation of this system. Thus far, the following aspects require further consideration throughout the development phase.

Privacy: since the proposed framework incorporates biometric recognition technologies, this involves the use of an individual's characteristics. Those data

are considered personal and sensitive and this raises issues related to biometric security and privacy. For that reason, storing and transferring a subject's biometrics must be achieved in a manner that minimises the threat to the interception and misuse of that information.

Scalability: developing a system that continuously generates data and stores and transfers them to a central data server introduces several challenges. This includes traffic management and synchronisation where optimisation is needed.

Storage: capturing, generating, and storing data raises technical and conceptual challenges. In particular, as the proposed system generates and collects more substantial amounts of data, this will require more investigation to enhance the system and adapt the storage issue.

Attack vectors: several threats are faced by the proposed framework, which include object manipulation (most of which are discussed in the experimental chapters 6, 7 and 8). It is, therefore, necessary to proceed to investigate and identifying attack vectors and security concerns and subsequently consider mitigation techniques in order to counter any potentially significant threats, as well as to evaluate the robustness and efficiency of the proposed system against such threats.

5.5.1 Research Questions Identified

Having introduced a system for the proactive biometric imprinting of digital objects, this research needs to investigate further how efficient and robust the proposed system is in inextricably linking the use of information (evidence) to the individual

users who access and use it. The following research questions are examined and investigated in the following chapters:

1. What is the feasibility of developing a biometric-based null-ciphered technique using digital images to better understand the robustness of hiding and successfully recovering biometric information under different levels of modification attacks to digital images?
2. What is the feasibility of developing a biometric-based grille-ciphered technique using digital images to better understand the robustness of mapping and successfully recovering biometric information under different levels of modification attacks to digital images?
3. What is the feasibility of developing a biometric-based grille-ciphered technique using digital documents to better understand the robustness of mapping and successfully recovering biometric information under different levels of modification attacks to digital documents?
4. What are the potential limitations and challenges in deploying a system for the proactive biometric imprinting of digital objects?

In order to investigate scientifically and answer the above questions, a set of extensive experiments were conducted to identify the strengths and limitations of the proposed approach.

5.6 Conclusion

The proposed system for the proactive biometric imprinting of digital objects is a novel digital forensic approach that would enable investigators to link the use of information (e.g., images, documents, and emails) inextricably to the individual

users who use and access it, through the use of steganography and transparent biometrics. The system utilises the computer hardware available (i.e., camera, keyboard, and mouse) to monitor an individual's interactions transparently and continuously by utilising different biometric techniques. However, as raised in this chapter, several considerations need to be taken into account of the privacy, efficiency, and security aspects introduced by the proposed framework.

6 Investigation of a Biometric-Based Null Cipher Using Images

This chapter investigates the feasibility of embedding individuals' biometric information inside digital objects (i.e., images) by utilising a null-ciphered technique, such as the LSB. The chapter includes an evaluation of the approach developed by measuring the success rate of retrieving the embedded signals under different types of attacks, such as cropping part of the image using different ratios. The chapter concludes with discussion and conclusion sections that examine the findings, along with a comparison with the previous research, and then highlights the limitations identified in the proposed approach.

6.1 Introduction

As explored in chapter 3, the detection of insiders' malicious activities is an important task. However, tracking and tracing such activities back to the source of the leakage are also key elements in preventing future incidents and seeking compensation. Linking the leaked data after they have been made public to the insiders concerned is not a simple task. Even if useful information were successfully mined from server and network logs, the sharing of authentication credentials and the Trojan defence are all too frequently used to mitigate the evidence.

Biometrics is a mean of identifying people by making use of someone's individual physical and behavioural characteristics, enabling a stronger association between the users and the IT system they are using (Ashbourn, 2015). Extending

the use of biometrics into transparent capture further enhances usability and security. Usability is enhanced as samples are captured without explicit or intrusive interaction with the user, merely capturing samples as the user regularly interacts with the system. Security is enhanced because, rather than having a single sample, the system is able to capture samples continuously from a range of biometric modalities, providing a far stronger approach than usual (Clarke, 2011). This could include capturing someone's facial image while that individual usually interacts with a computer, using, for example, the embedded web camera.

Furthermore, multimodal transparent biometrics could be employed that would make circumvention and forgery that much more challenging (Ceccarelli *et al.*, 2015). More closely aligning users' biometric information to the data with which they are interacting could lead to more reliable and timely attribution of activities. Therefore, the next section provides an investigation into a biometric-based null cipher using images.

6.2 Methodological Approach

This investigation proposes a secure and portable approach that can link individuals with the object with which they interacted (e.g., digital images) without the need for a centralised database that stores interaction logs or the presence of the original object. The correlation between the user and the digital object is established by leveraging biometric signals, such as distinctive facial features, transparently (so the user is neither inconvenienced nor aware when samples are captured, which will mitigate forgery) with a steganographic technique for embedding them into the object. It is essential to state that the proposed algorithm

is not intended to develop a new data-hiding technique but rather to investigate whether the biometric signal(s) can still be recovered after significant modification of the original digital object.

The main components of the proposed approach are built around the capturing, processing and classification of biometric signals derived from built-in sensors within the computing technology that can be captured without explicit interaction by the user. The transparent, continuous and multimodal capture of biometric signals is a key method in mitigating against forgery (Clarke *et al.*, 2017). As illustrated in Figure 5.1, the system elements and the main process flow form critical parts of the imprint process. The design of the embedding algorithm also takes into consideration the following aspects: the effectiveness of the system, the security of the biometric information, and robustness against potential attacks, such as cropping and modification of the image. The embedding procedure, as illustrated in Figure 5.3, follows the steps outlined below.

6.2.1 Capturing an Individual's Biometric Information

An individual's biometric signals are transparently and continuously captured while the user interacts with the object. For instance, when the user edits/writes a particular document, different biometric modalities could be used to profile the subject, including facial recognition and keystroke dynamics. A camera fitted in a computer can capture facial samples while the user is looking at the computer screen. In addition, keystroke dynamics can be used to identify individuals by the way they type, which can also be analysed and processed to provide a discriminative biometric signal.

6.2.2 Extracting a Biometric Signal

The biometric signal is then extracted and forms the biometric feature vector(s). The size and number of dimensions of the vector vary—depending upon the algorithm that is used for detecting and extracting the signals for the given biometric modality. For instance, a typical facial feature vector used in this study is 60 digits long.

6.2.3 Transferring the Biometric Signal and Image into Byte Streams

For most spatial domain image steganography systems, the first step in the process is to transfer the representation of the data (that is to be embedded) along with the carrier into byte streams. Only those pixel bits selected are converted into a bit matrix and substituted with the payload data with respect to the steganographic algorithm used.

6.2.4 Padding the Flag and the Timestamp

In order to maximise the anonymity of the embedded data, a flag of 10 alphabet characters in length is randomly generated for each print and padded to the beginning of the signal to be used later as a lead for the extraction algorithm to identify the locations of the embedded data. The length of this generated token ensures high entropy, as no two prints could hold the same token. A timestamp in the Epoch time format (also known as POSIX time) is inserted after the padded characters (flag) to provide more intelligence information to the investigator, as it indicates when exactly the user interacted with the object that is being examined. In the recovery process, the flag bytes are then decrypted one by one, looking for ten consecutive alphabet characters (the flag). This helps to ensure the number

of spurious keys is zero and thus the information that follows is a viable timestamp and biometric feature vector.

6.2.5 Choosing the Embedding Locations

The number of times (prints) that the payload is injected into the image can be specified as a variable within the embedding algorithm. Hiding the tracing information in multiple locations increases the likelihood of finding the hidden data after the image is modified, for instance by cropping part of the image. Therefore, the coordinates of the pixels (xi, yi) are randomly selected by the embedding algorithm to specify where the payload is injected.

6.2.6 Encrypting the Payload

Prior to the embedding process, the data are encrypted using AES. AES is symmetric encryption and the encryption keys (K*i*) are generated based upon a predefined master key. The keys are used to encrypt each print of the payload so that the inserted data are not alike, in contrast with a single key being used to encrypt all the prints. This ensures that the imprints (payloads) have a different pattern and detecting those data can be harder. If the same key is used to encrypt the data, the resulting ciphers will be identical; thus, finding those payloads is simply a matter of searching for identical values across the carrier object.

6.2.7 Embedding the Encrypted Data into the Image

The embedding function inserts the encrypted byte streams into the image. The maximum embedding capacity varies depending on the given image size, colour depth and steganography method. For instance, within true colour images, the

number of elements in the cover, *n*, is three times larger than those in greyscale images.

Figure 6.1 illustrates the concept of how the bits in the original carrier are replaced when using the LSB method to hide data. Each bit in the payload is substituted for the LSB in the original data.

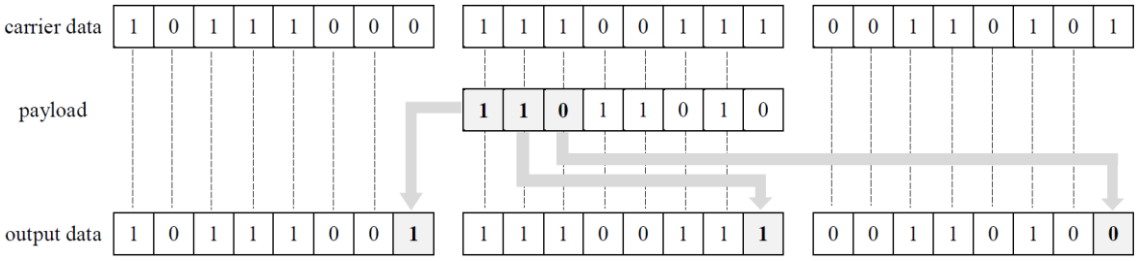

Figure 6.1: Applying LSB using one least significant bit

The resulting image conceals the biometric signal(s), flag and timestamp. A substantial advantage of this system is that the embedding process is entirely random, which makes predicting where the data are hidden highly challenging. To recover the embedded information, the extraction algorithm works in reverse when compared with the embedding algorithm. It takes the image in question and the decryption keys as inputs; the keys are the same as those used by the embedding algorithm to encrypt the payload. Since the location of the pixels in which the embedded data are located are selected randomly, the extraction process starts at the first pixel (0, 0) and continues until the last (x_n, y_n) (i.e., all possible locations). All the interesting bits are mapped out into a matrix of byte streams.

The following steps explain the process of extracting the embedded data from the examined object:

1. Read the examined image.
2. Find the hidden flag that indicates the location of the secret data.
3. Continue the process until the message is fully extracted from the image.
4. Split the extracted message into parts (flag and biometric signal).
5. Use the encryption key to decrypt the ciphertext to find the original information.
6. Reconstruct the biometric and supplemental information.

The bytes are then decrypted one by one, looking for 10 consecutive alphabet characters (the flag). Once the flag has been found, the 70 bytes (131 bytes when two biometric signals were embedded) next to the flag are written out in plaintext format, which illustrates the embedded payload that contains the timestamp as well as the biometric signal(s).

6.3 Experimental Analysis

This research aims to investigate and better understand the nature of hiding and successfully recovering biometric information under different levels of modification attack to the digital object in a way that could lead to the individual who interacted with the digital files (images in this experiment). In order to achieve this, a real facial biometric vector was captured for one individual using a developed script while the individual was interacting naturally with the computer. Two variations of the payloads were examined. The first variation consists of only

a single biometric feature vector (facial biometric) along with the flag and timestamp. The Fisherface algorithm was used to generate the feature vectors for the facial images captured from the user (Belhumeur, Hespanha and Kriegman, 1997). Fisherface projects the image set to a lower-dimensional space so that the resulting within-class scatter matrix *SW* is non-singular. This is achieved by using PCA to reduce the dimension of the feature space to *N* - *c*, and then applying the standard Fisher's Linear Discriminant (FLD) to reduce the dimension to *c* - 1. The optimal projection ($W\ opt$) is given by:

$$W_{opt}^{T} = W_{fld}^{T}\, W_{pca}^{T}$$

Where

$$Wpca = \arg\max_{w} |\, W^{T} S_{T}\, W\, |$$

$$Wfld = \arg\max_{w} \frac{|W^{T} W_{pca}^{T} S_{B}\, W_{pca} W|}{|W^{T} W_{pca}^{T} S_{W}\, W_{pca} W|}$$

Where $S_B$the between-class scatter matrix is be defined as:

$$SB = \sum_{i=1}^{c} N_i\ (\mu_i - \mu)(\mu_i - \mu)^{T}$$

Where $S_w$the within-class scatter matrix is be defined as:

$$S_W = \sum_{i=1}^{c} \sum_{x_k \in X_i} (x_k - \mu_i)(\mu_k - \mu_i)^{T}$$

Where mi is the mean image of class X_i , and N_i is the number of samples in class X_i.

The optimization for $W\ pca$ is performed over n × (N - c) matrices with orthonormal columns, while the optimization for $W\ fld$ is performed over (N - c) × m matrices with orthonormal columns. In $W\ pca$, the smallest c - 1 principal component is ignored.

The resulting vector is four-dimensional and 60 digits long (zeros are padded to the beginning of a vector whenever it has fewer than 60 digits). The cipher stream of the payload is 80 bytes long. The second variation combines two biometric signals (e.g., facial and keystrokes) and, along with the flag and timestamp, is 141 bytes. The Lena picture (a standard test image widely used in the field of image processing) was used as the test image in this study.

The minimum pixel size needed to carry out at least one imprint of a single biometric modality is 214 pixels for RGB images and 1,128 pixels for greyscale images when the LSB approach is used for embedding. To determine the maximum number of prints that a test image can accommodate, when only the LSB per colour is substituted, the total number of LSBs of the given image is divided by the size of the payload (in bits).

Table 6.1 shows the maximum number of prints (times) that the payload can be embedded in the test image for the various image sizes.

Table 6.1: Maximum number of prints that can be embedded into a given RGB image

Image size (Pixels)	Max. number of prints using all available LSBs	
	Single bio-signal	Two bio-signals
256x256	307	174
512x512	1,220	696
1,024x1,024	4,880	2,784
2,048x2,048	19,520	11,136

By using this steganographic approach, in which the biometric signals are directly inserted into the image, the quality of the original object changes depending upon the size of the embedded data. Therefore, to measure the error rate and noise introduced by the embedding algorithm, both the mean squared error (MSE) and peak signal-to-noise ratio (PSNR) are calculated. MSE is a metric that estimates the quality of the modified image compared with its reference (original) based on the squared difference between the two. The MSE value for the original image (x) and the modified version (y) is computed using Equation 3:

Equation 3: Mean squared error

$$MSE(x,y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2 \qquad \text{Equation 3}$$

The MSE is used to compute the PSNR using a logarithm (dB, as shown in Equation 4:

Equation 4: Peak signal-to-noise ratio

$$PSNR = 10\,log_{10}\frac{L^2}{MSE} \qquad \text{Equation 4}$$

Where L represents the dynamic range of possible image pixel intensities. For instance, for images that have allocations of 8 bits per pixel of greyscale, L is computed by 256 − 1 = 255. Figure 6.2 illustrates the effect of embedding payloads into an image (512x512 pixels) measured by MSE and PSNR. It is generally known and expected that, as the size of the embedding data increases (the number of prints in this case), the error rate and noise introduced to the image will also rise.

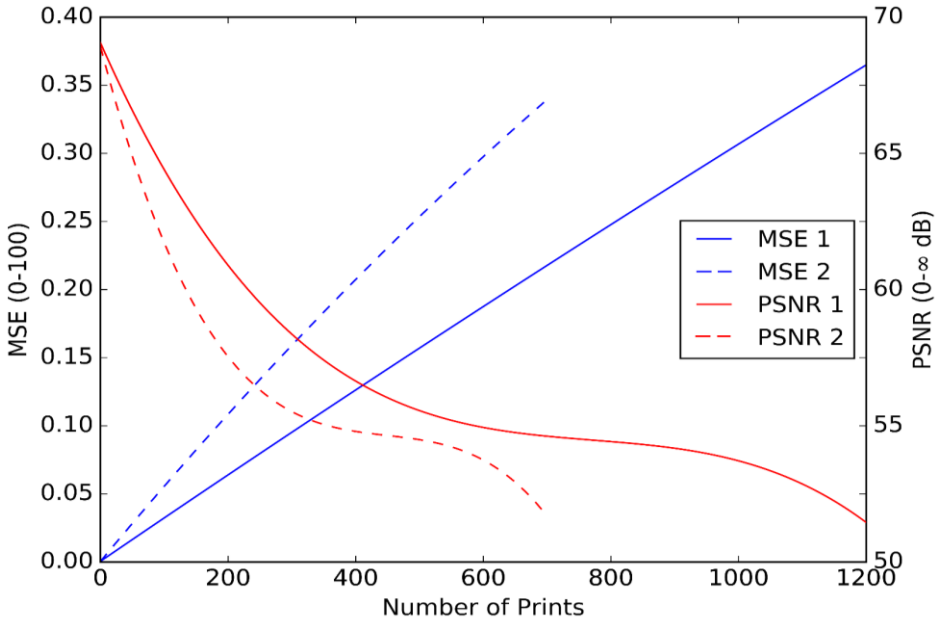

Figure 6.2: MSE1 and PSNR1: single biometric signal per print; MSE2 and PSNR2: two biometric signals per print

The watermarked images are then randomly cropped—to ensure that the result is more generalised—by 25%, 50% and 75% of their original size (as Figure 6.3 illustrates). The cropping attack simulates a case in which only part of an imprinted image is available for examination. Such an attack could be performed by a malicious user to wipe any traces that could identify the source of the leakage.

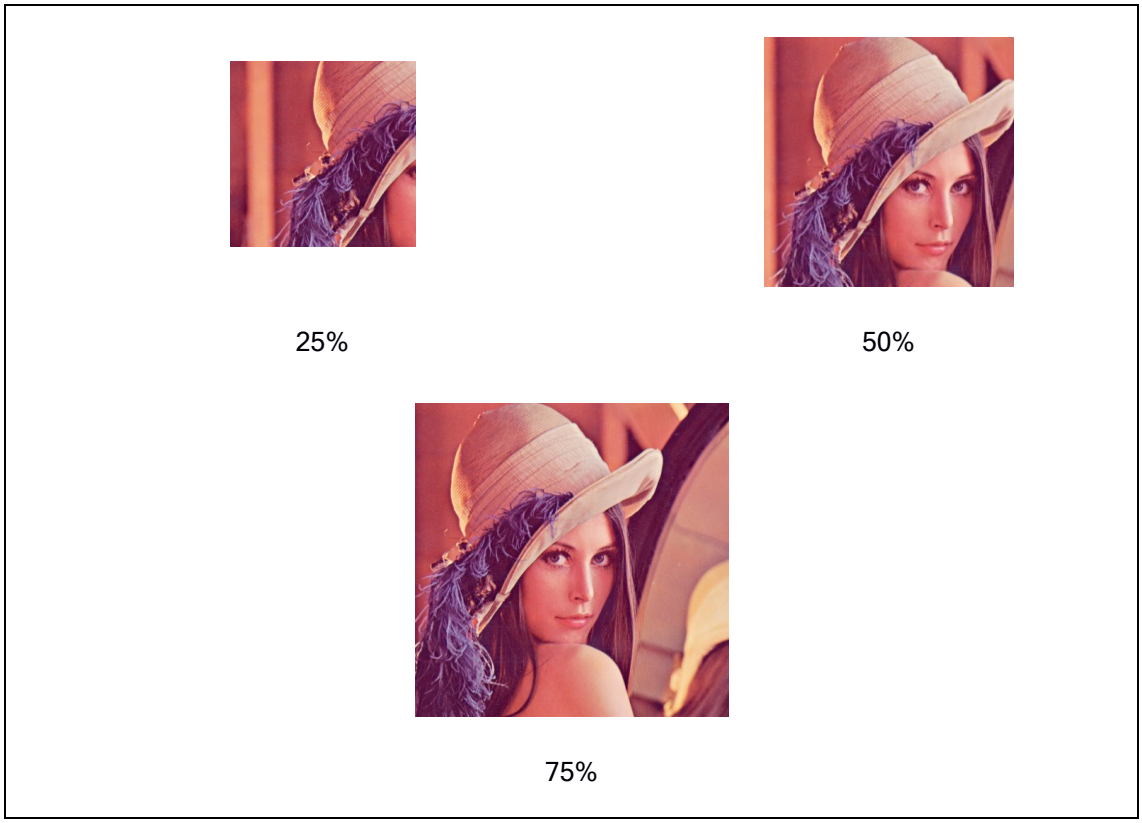

Figure 6.3: Cropped versions of the test image

Figure 6.4 illustrates the number of recovered prints under the three cropping ratios (25%, 50% and 75%), along with the different number of embedded prints per image. It is clear that, as the size of the examined watermarked image increases, the number of prints being extracted also rises. The result for the single biometric signal per print experiment shows that even with only 25% of the watermarked image, it is still possible to retrieve and recover the embedded data

successfully by searching for the encrypted flag. Therefore, the number of recovered prints is significantly correlated with the number of embedded prints, as well as the type of modification being performed on the image.

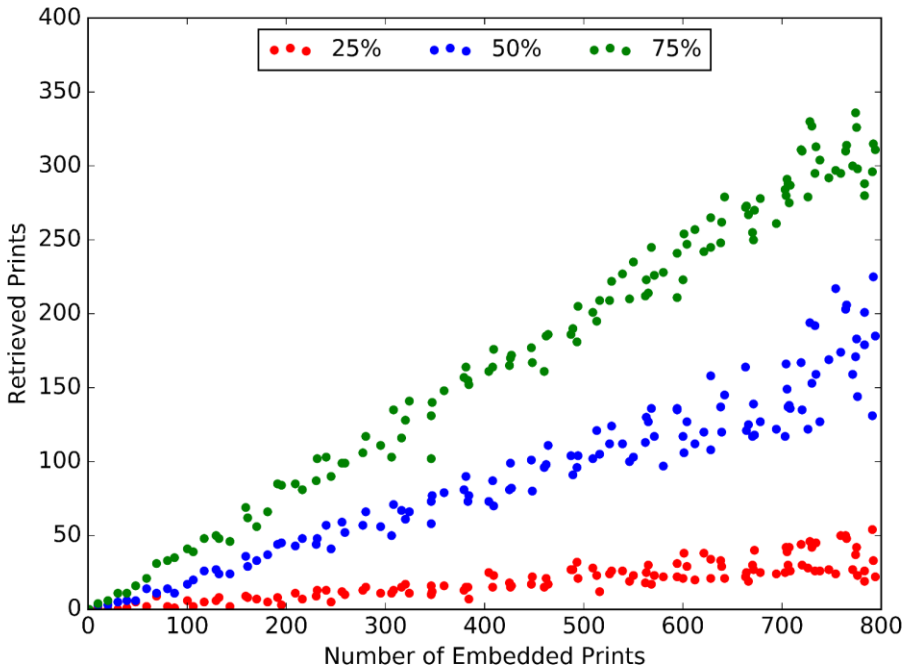

Figure 6.4: Number of prints extracted for a single biometric signal per print

Furthermore, two biometric signals per print were experimentally examined to investigate whether the length of the embedded data could affect the number of recovered prints. Figure 6.5 demonstrates the number of recovered prints when the watermarked image is cropped by 75%.

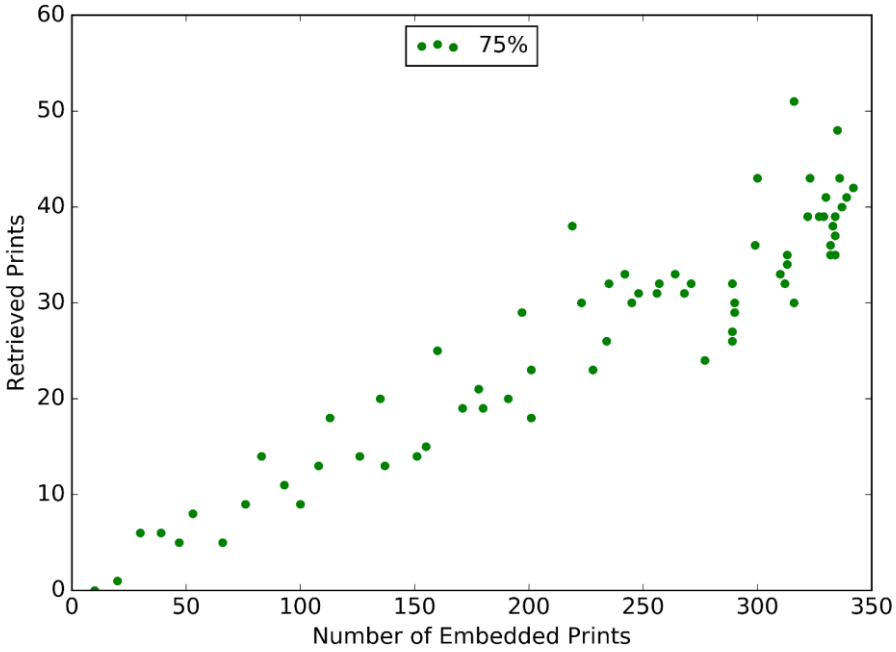

Figure 6.5: Number of extracted prints for two biometric signals per print

By comparing the results obtained with the experiment above, it is notable that the number of recovered prints has dropped significantly due to the increase in payload size. The size of the payload that contains two signals is 141 bytes (1,128 bits), which requires 376 pixels to accommodate (when 1 bit per colour is substituted). Whereas, the payload that contains a single biometric signal is 80 bytes long, which uses 214 pixels of the watermarked image. The difference between the two variations is ~76% in terms of size. Therefore, due to this difference, cropping a watermarked image that carries two signals per print by less than 75% (i.e., 50% or 25%) is highly likely to render extracting the embedded prints unfeasible. However, it is still possible to retrieve at least one signal, which is sufficient for the ultimate aim (i.e., having a biometric signal that could indicate a particular individual).

Image size ratio to its orginal									
100%	90%	80%	70%	60%	50%	40%	30%	20%	10%
512	486	458	428	397	362	324	280	229	162
768	729	687	643	595	543	486	421	343	243
1,024	971	916	857	793	724	648	561	458	324
1,280	1,214	1,145	1,071	991	905	810	701	572	405
1,536	1,457	1,374	1,285	1,190	1,086	971	841	687	486
1,792	1,700	1,603	1,499	1,388	1,267	1,133	982	801	567
2,048	1,943	1,832	1,713	1,586	1,448	1,295	1,122	916	648

Figure Keys
0 signal
1 signal
2 signals
3 signals
4 signals

Figure 6.6: Correlation between the number of signals per print, size of the image and the likelihood of recovery

To further demonstrate the correlation between the number of signals per print/ payload, size of the image and the likelihood of the successful recovery of the embedded data, Figure 6.6 shows that the correlation between those variables is a direct relation. The colour indicates the recovery likelihood: dark red expresses a low chance and dark green implies a high chance of recovery. The circles represent the number of signals per print/payload. From this data, it can be seen that, as the size of the image increases, the number of signals per print also rises, as does the likelihood of recovering the embedded prints.

6.4 Discussion

From the results obtained, it is clear that the use of the LSB approach, especially under a modification attack (i.e., cropping the image), enables the embedding and recovery of biometric information. Although embedding the payload directly into the image when using the null cipher approach introduces noise to the object, it does not require the original image to be present or the existence of a database

to store metadata or signatures. Whereas, a grille cipher stores the generated digital imprint file in a centralised database. This file represents the locations within the object, each of which corresponds to a respective portion of the extracted biometric features.

That said, some several limitations and challenges exist and require further research, these include:

- The placement of the bits to be embedded could be randomised using a random function (e.g., scattering) so that it would be almost impossible to retrieve the embedded data without knowing the seed for the random function. Although this is a useful approach, it only works if the image has not been modified or cropped after the data were embedded.

- Using the LSB for inserting the payload is vulnerable to file type transformation. For instance, transforming the resulting image (e.g., PNG) into other formats (e.g., JPEG) is highly likely to destroy the embedded data by using the compression algorithm.

- The proposed algorithm uses a conventional spatial technique (LSB substitution) that embeds the data directly in the intensity of the pixels. In other steganographic approaches, such as frequency techniques, the images are first transformed into an intermediate image and then the data are embedded in the image. A common method in the frequency domain is to modify the relative size of two or more DCT coefficients in an image block, embedding one bit of information in each block. The algorithm should be robust against JPEG

compression, so DCT coefficients with equal quantisation values should be chosen, in accordance with the JPEG quantisation table (Soria-Lorente and Berres, 2017).

Further investigation and experimentation are needed into attacks targeting LSB techniques, such as the Chi-square and the extended Chi-square attacks, and the Zhang and Ping Method, which targets the Jsteg-like algorithm (Fridrich, 2004). This study has focused only upon the image file type and differing file types have varying degrees of stability due to their structure. For example, Word documents and their underlying data structure can change considerably, given small alterations to a file. Therefore, alternative approaches need to be developed to overcome this issue.

6.5 Conclusion

This chapter has examined the feasibility of embedding an individual's biometric signals into image files by utilising a null-ciphered technique, the LSB steganography approach. The proposed algorithm takes the biometric signals as a feature vector along with the supplementary information (flag and timestamp) and embeds them into the interacted image after the payload is encrypted. The experimental results have shown that it is possible to retrieve the biometric signals successfully even when the image is cropped to 25% of its original.

The results obtained are robust against many of the scenarios examined, and the nature of conventional watermarking and null cipher-based processes is not to modify the digital object in a manner that is noticeable. It does, nonetheless,

modify the digital object. There may be situations in which this modification is not desirable; for instance, when preserving the integrity of the object is crucial, as is the case in a digital forensic investigation. Directly performing modifications to the object, beyond the computer user's awareness, would change the integrity of the file. Hence, this type of evidence of a crime would not be accepted in a court of law. The next chapter proposes a novel approach that does not directly modify or change the examined object and preserves its integrity. Instead, it generates a correlation file and stores it in a local database to be used later when needed.

7 Investigation into a Biometric-Based Grille Cipher Using Images

This chapter presents an investigation into a biometric-based grille cipher using images by linking a subject (i.e., computer user) with an object of interest (e.g., image) and using the individual's biometric sample, such as a facial biometric, without modifying the object being imprinted. The chapter begins by introducing the proposed approach, including the core process, followed by an explanation of the experimental methodology of using different types of attack vectors to evaluate the robustness of the proposed method. The experimental results are then presented, followed by a discussion of the findings and the limitations identified from the experimental analysis.

7.1 Introduction

Unlike most existing methods, such as digital watermarking or null ciphers, in which the integrity of the object is modified (Charbonneau and Simon, 2014; Nelson and Xie, 2014), a generic grille cipher approach employs a template that is used to cover the carrier message; the words that appear in the openings of the template are the hidden message. Unfortunately, designing a cipher scheme that satisfies all the constraints remains a difficult problem. The approach proposed in this chapter 'imprints' the object with any given data (i.e., the user's biometric feature vector). Therefore, the imprinting process employed can be described as a correlation of the feature vector with the object.

7.2 Methodological Approach

The proposed framework consists of two engines: a biometric engine and a grille cipher engine. The biometric engine transparently captures and extracts the user's biometric samples (e.g., facial features, keystroke analysis, behavioural profiling) and stores them in a database on the user's computer. The grille cipher engine retrieves the object metadata and its hexadecimal (hex) representations and requests the latest user's biometric feature vector from the biometric engine for use in the imprinting process. Finally, the imprints generated are stored in a centralised database for later analysis when required. Figure. 7.1 illustrates the framework architecture for the proactive biometric-enabled forensic imprinting system. Upon detection of data leakage, the object (whether it is posted on a public website or captured by the network) can be analysed for the biometric imprint.

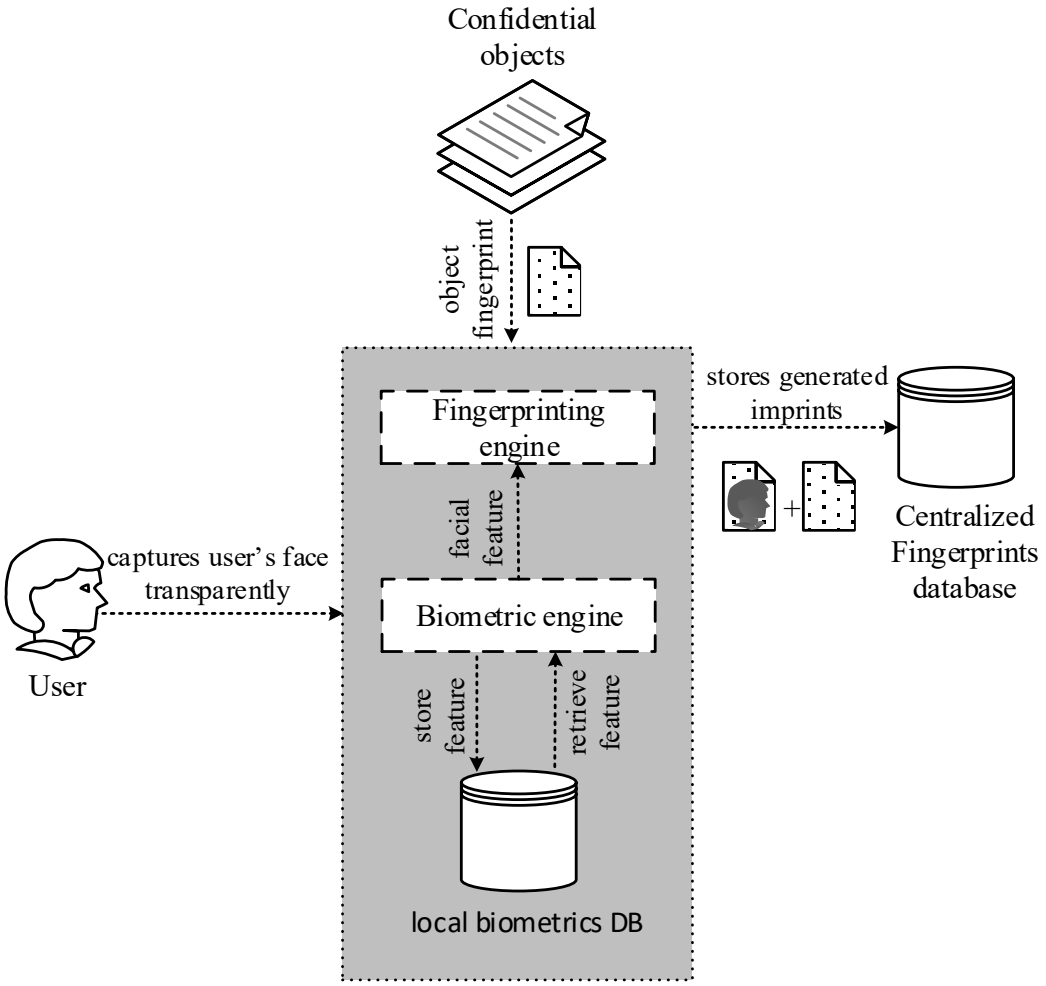

Figure. 7.1: The proposed framework architecture

The sample is extracted and then processed by a biometric system in order to determine the last user to interact with the object. The generation process of the imprints is inspired by the benefits of employing the grille cipher technique. Grille ciphers were used in the past (prior to the modern null cipher) as a means of transferring/exchanging secret messages between two parties. They were initially used to extract hidden messages from a plain text by mapping the text through a pierced sheet, such as card. For instance, the words “secret” and “plan” can be

extracted from a letter puzzle by applying a piece of cardboard with appropriate apertures that map the desired locations of the letters, as Figure 7.2 illustrates.

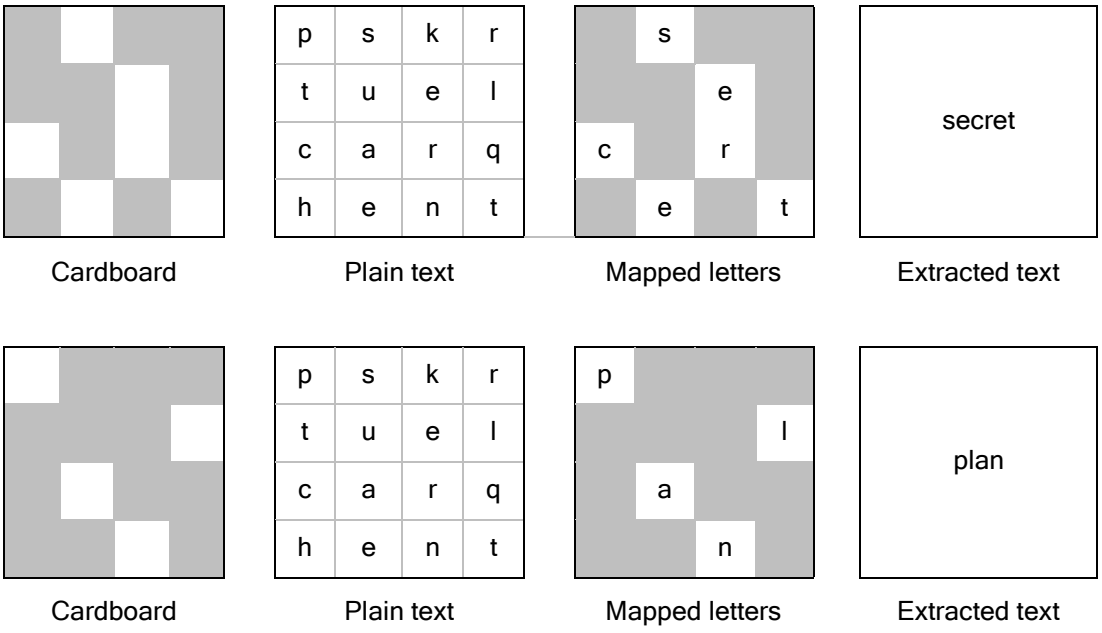

Figure 7.2: Grille cipher mapping example

The embedded secret message can be retrieved by mapping specific locations. Hence, applying the same technique to imprint a biometric feature vector to an object file is possible, as the object can be an image file, document, video, or any other digital file type. Adapting the grille cipher technique to the proposed approach involves several consecutive steps, which are described below.

7.2.1 Preparation of the Feature Vector and Object

The preparation step converts both the feature vector and object into their hex representations for mapping purposes. The index of each character is preserved during this conversion, which begins with zero for the first character and continues in ascending order until the last. The process of conversion is not necessarily achieved by transforming each character, since reading the whole object in binary

mode allows for low-level representations of both hex and binary. Character-by-character (or byte-by-byte) indexing is still required in order to generate the object index list.

7.2.2 Mapping the Feature Vector to the Object

After obtaining the hex representations of the feature vector and the object, each hex value in the feature vector is mapped to its equivalent position in the object's hexes to retrieve possible positions where they both match. Accordingly, the mapping process returns lists of indexes of the matched hexes.

7.2.3 Generating the Feature Vector Imprints

By retrieving the positions of each character of the feature vector with the object, it is now possible to generate imprints based on the list of indexes, which means that multiple imprints of the whole feature vector can be generated by combining those positions.

The pseudocode of the imprinting process, starting from the preparation is illustrated below in Algorithm 1.

Algorithm 1: Imprinting algorithm

Input: Feature Vector (*FV*), Object (*O*)

Output: Imprints

```
1:  function PREP (FV, O)
2:      for each value in FV & O:
3:          Convert FV, O into its HEX representations
4:          Retrieve the index of each value
5:      Return FV_HEX, index, O_HEX, index
6:  function MAPPING (FV_HEX, index, O_HEX, index)
7:      for each value in FV_HEX, O_HEX:
8:          index(O_index) ← FV_HEX ∩ O_HEX
9:      Return index(O_index)
10: function IMPRINTING (indexes)
11:     imprint ← Combine unique indexes from the
12:     retrieved index list
13:     Return imprints
```

The following example explains how the imprinting algorithm works in practice. For demonstration purposes, assume that the following feature vector needs to be mapped to an object, as presented in Figure 7.3.

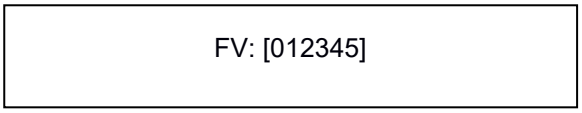

Figure 7.3: Feature vector and an object

FV	**0**	**1**	**2**	**3**	**4**	**5**
Hex(FV)	30	31	32	33	34	35
Hex(FV)index	0	1	2	3	4	5

O	**0**	**1**	**2**	**3**	**4**	**5**	**0**	**1**	**2**	**3**	**4**	**5**	**0**	**1**	**2**	**3**	**4**	**5**
Hex(O)	30	31	32	33	34	35	30	31	32	33	34	35	30	31	32	33	34	35
Hex(O)index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Figure 7.4: Hex representation of the feature vector and the object

The process occurs regardless of the file type of the object since any file type can be transformed and treated as a hex representation. The first step in the proposed algorithm is to convert both the feature vector and the object into their hex representations. In accordance with the ASCII table, Figure 7.4 shows the converted characters as the hex alongside the position of each value (index). In this example, each value of the feature vector exists in more than one location within the object. For example, “30” (the hex representation of zero) is located in positions 0, 6, and 12 (as show in Table 7.1). In the same manner, the mapping process continues for all subsequent feature vector values until all possible positions are retrieved. In addition, the column titled “*positions in the object*” presents the retrieved positions for each value of the feature vector.

Table 7.1: Feature vector value positions in the object

Original value	Hex representation	Positions in the object
0	30	0, 6, 12
1	31	1, 7, 13
2	32	2, 8, 14
3	33	3, 9, 15
4	34	4, 10, 16
5	35	5, 11, 17

The last step in this example is to generate all possible imprints from the positions retrieved. Since each feature vector value is located in three different locations, the total number of unique imprints that can be generated from these positions is three, as illustrated in Table 7.2. Therefore, using any value for these imprints, it is possible to reconstruct the original feature vector from the object by reversing the mapping process. After explaining how the imprinting technique works through the example given, the next section investigates the feasibility of imprinting biometric feature vectors with images and later recovering them (even after object modification).

Table 7.2: Possible imprints

Imprint number	Imprint
1	0, 1, 2, 3, 4, 5
2	6, 7, 8, 9, 10, 11
3	12, 13, 14, 15, 16, 17

7.3 Experimental Methodology

The primary goal of the experiment is to assess the feasibility of the proposed hypothesis of the subject's feature vector being forensically linked to and retrieved from an object of interest. Therefore, it is critical to evaluate its performance in a sophisticated, subject-related manner. A total of four experiments were conducted:

- The first experiment retrieved the feature vector from the original imprinted image.
- The second experiment examined a situation in which the image is modified in one area with an increasing proportion of modification.
- The third experiment verified the case in which an image is modified in several areas.
- The final experiment investigated what happens when only parts of the original image are available and the rest is missing.

The feature vector used in the experiments represented a real facial feature vector sample with a length of 57 numeric characters, as illustrated in Figure 7.5. The length of the vector relies upon the feature extraction algorithm used to compute the feature vector. In this study, the Fisherface algorithm was used to compute the feature vector for the captured user's facial images (Belhumeur, Hespanha and Kriegman, 1997). The algorithm also performs PCA and LDA for dimensionality reduction (Yu and Yang, 2001).

[1679.2235398, -1555.40390834, -1140.07728186, -1999.85500108]

Figure 7.5: Facial feature vector

With regard to the objects used in the experiments, UCID image dataset version 2 was used (Schaefer and Stich, 2003). The database contains a total of 1,300 images of two sizes: 1,234 x 1,858 or 1,858 x 1,234 width, height in pixels. For this study, only the first 100 images were used from the dataset, since it was assumed that this number would be enough for the purpose of evaluation. The implementation of the proposed algorithm was developed in Python due to its flexibility in terms of list comprehension and image processing. Moreover, Python's built-in library has several useful functions, such as *map* and *zip,* which facilitate a number of relevant operations.

7.3.1 Retrieving the Feature Vector from the Original Imprinted Image

The aim of this experiment was to imprint the feature vector as many times as possible with each image in the dataset. The first experiment examined the possibility of generating imprints between the feature vector and the object used. Since there is a high probability that the subject or another party (intentionally or unintentionally) could somehow modify the object in question after it is imprinted, the subsequent experiments investigated the accuracy of retrieving the feature vector from the object in several situations.

7.3.2 Modification in One Area

The second experiment evaluated the imprinting mechanism after the image was modified by a different percentage. The simulation of this was performed by randomly choosing a section of the image as a rectangular box of a growing size to reflect an increasing proportion of modification. Equation 5 was used to determine the size and random position of the modified section. The equation takes the following three variables:

- *w*: image width;
- *l*: image height;
- *s*: the desired modification percentage.

The equation gives four values: x and y are random values between zero (image width) and zero (image height), respectively. These set the top-left pixel position of the modified rectangle (as highlighted in red in Figure 7.6); the third and fourth values are for the bottom-right corner of the rectangle (shown in blue).

Equation 5: Modification size

$$P_{(w,l,s)} = \sum_{x=0}^{w-1} \sum_{y=0}^{l-1} (x, y, \ x + \frac{w}{10 \cdot \sqrt{s}}, y + \frac{l}{10 \cdot \sqrt{s}})$$

Figure 7.6: Sample of a modified area of an image

In this experiment, the imprinted images were modified by 5% increments, which means that the first alteration rate was 5%, then 10%, 15% and so forth until reaching 100%. Figure 7.7 demonstrates samples of an image modified by different rates. The (a) image is modified by 5% of its original size and the other three samples (b, c and d) are modified at rates of 35%, 65%, and 95%, respectively.

Figure 7.7: Sample of a modified part of an image

7.3.3 Modification in Multiple Areas

The third experiment was similar to the previous one, except that the modifications occurred in several parts of the image instead of an increasing proportion of one area. This type of attack is more influential since various and random parts of an image are affected by these alterations. In order to simulate this type of modification, the dataset images were altered using multiple rectangular boxes, each of which was equal to 1% of the total image size. Therefore, simulating a 5% alteration of random locations would need five of these boxes in an image. This experiment also assessed the proposed technique

with object alteration in 5% increments of the original size. Figure 7.8 illustrates four sample images modified by 5%, 35%, 65%, and 95%, respectively.

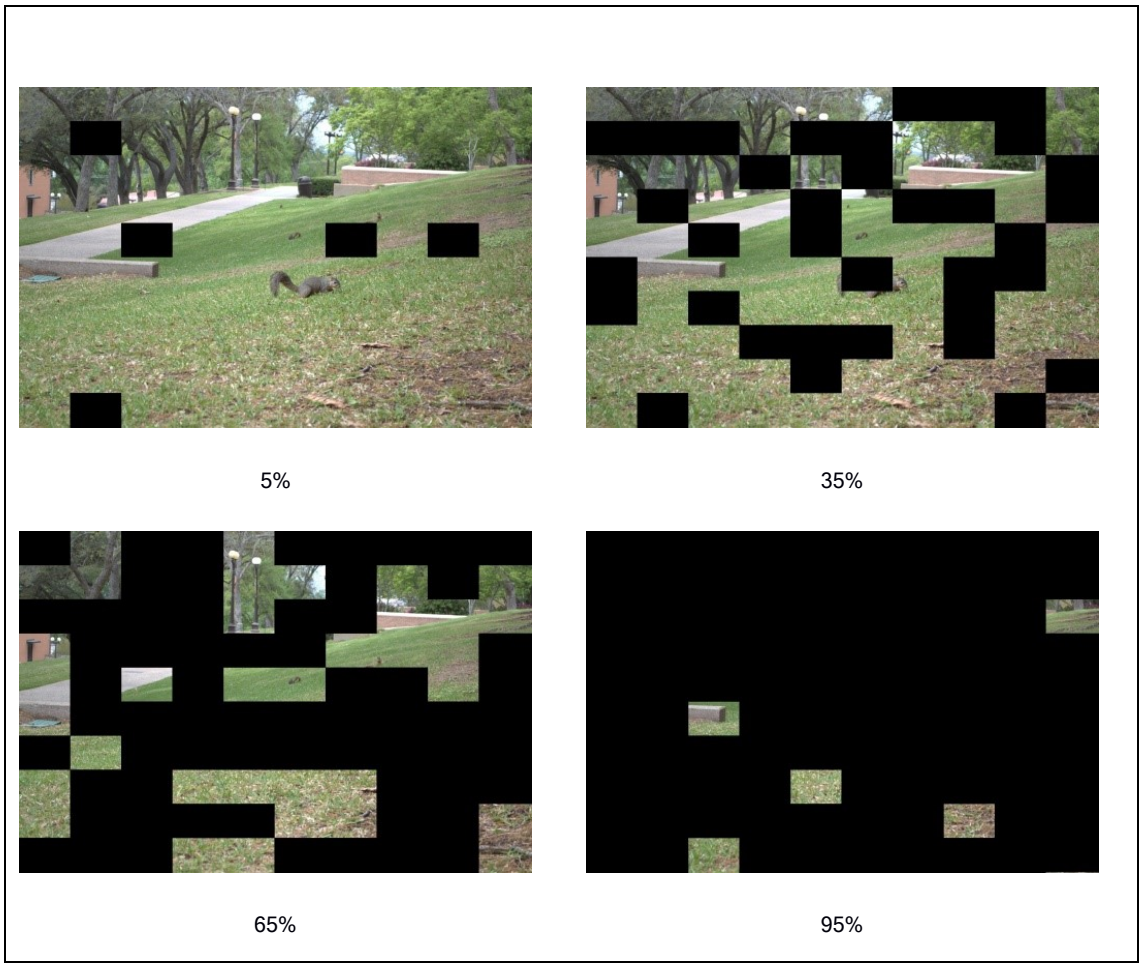

Figure 7.8: Sample of a modified area of an image

7.3.4 Partial Image

Further investigation was needed to better understand the effects of different attack vectors on retrieving the imprinted feature vector. Therefore, the last experiment presented in this chapter is interesting in terms of the results obtained. The fourth experiment simulated a scenario in which only part of the imprinted image was available and the rest was missing; for instance, the imprinted image might have been resized or cropped. To simulate this type of alteration, a random

section of the images in the dataset was cropped to different sizes, starting from 5% of the original size; then, in each subsequent test, random sections were again cropped at increments of 5%. Figure 7.9 illustrates some of the cropped samples.

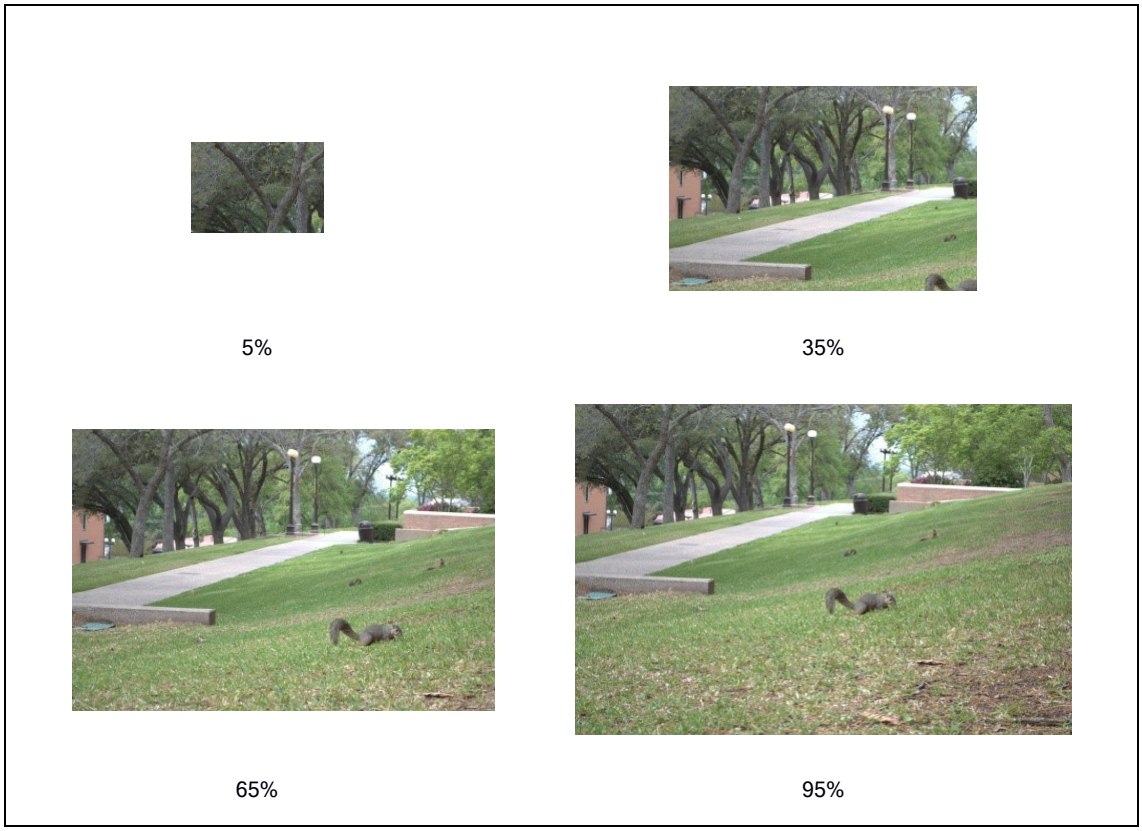

Figure 7.9: Samples of an image cropped to certain percentages

7.4 Experimental Analysis

The results of experiment one show that the average number of generated imprints was 854 per image. The minimum number of imprints in a single image was 244 and the maximum 1,815. This suggests that the mapped feature vector could be retrieved and reconstructed from any of the imprints. The number of imprints achieved is not surprising since the feature vector always contains

numerical values (0-9). Therefore, there are many matches between the feature vector and the hex values of the images. A reconstruction of the feature vector from the unmodified images was also possible using those generated imprints with an accuracy of 100%. This was achieved by reversing the imprinting processes.

In the second experiment, it was found that the imprinting technique was effective since the imprinted feature vector was successfully retrieved from an average of 97 out of 100 images, even when the modification percentage was 80%, as Figure 7.10 illustrates. On average, it takes around 3 milliseconds to generate an imprint (with an average size of imprints of 472 bytes). However, after a modification of 80% to the images, the number of valid retrieved feature vectors significantly drops due to the loss of most of the imprint values across those images. It can be seen that there is a fluctuation in the percentage of retrieved feature vectors while the modification rate increases. It worths to mention that if at least one imprint per image is successfully retrieved (with a minimum number of imprints in a single image was 244 and the maximum 1,815), the result is counted valid. Now, the fluctuation in the result has occurred as the nature of these imprints are scattered among the image, and due to the randomness of the attack (modifying part of the image in random locations), this could change the values of the mapped offsets/indices while in another rate of modification, although it is still restoring those mapped imprints, some of which are not affected. While after a massive amount of change, a decline occurs because the critical set of mapped index values changes after such a high modification rate. However, it was clearly

illustrated that it is feasible to reconstruct the feature vector from the imprinted objects even after significant destruction of its original values.

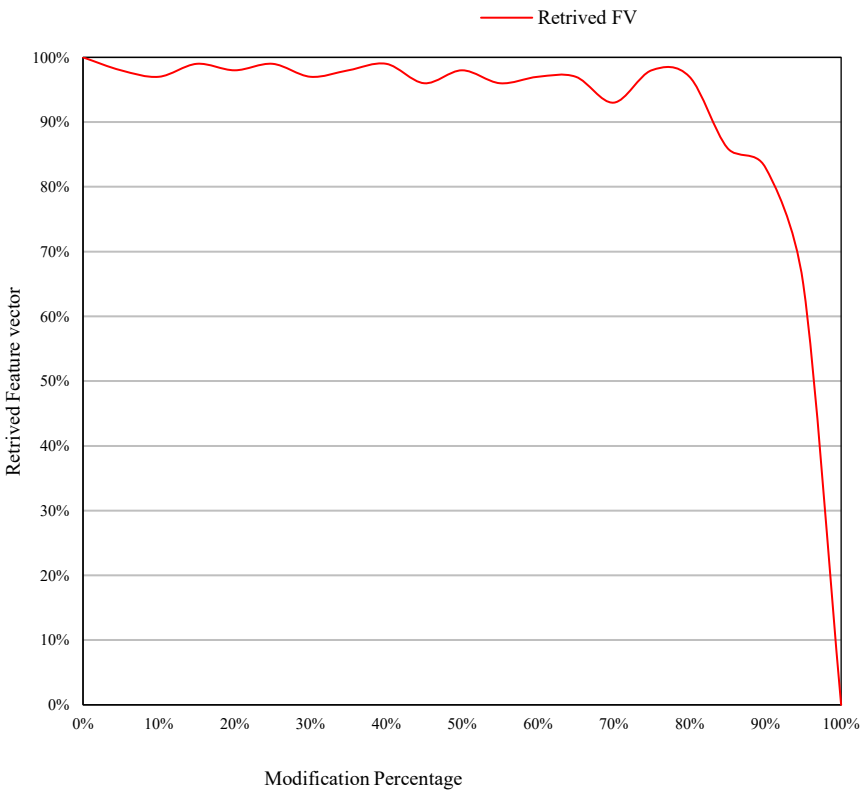

Figure 7.10: One area modification attack

In the third experiment, in which the modification took place in multiple areas, the results show that the imprinted feature vector was successfully retrieved, even when the images were altered in a more sophisticated way than in the one area modification attack (experiment two). Figure 7.11 shows the percentage of images for which the feature vector was successfully retrieved.

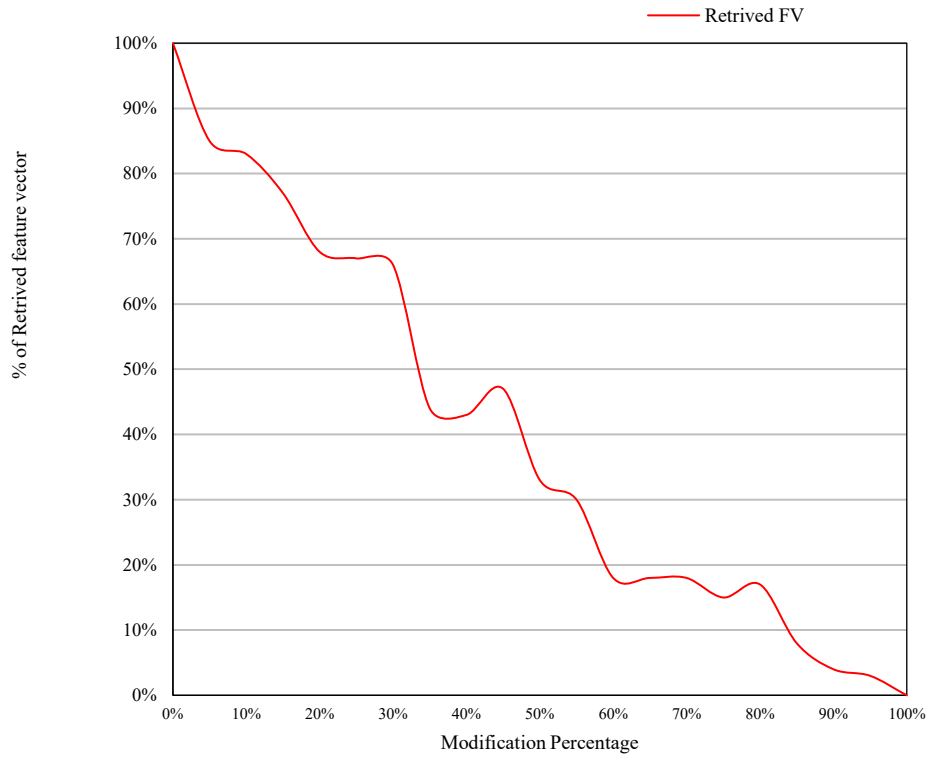

Figure 7.11: Multiple parts modification attack

Changing the values of specific pixels by printing the black boxes after the imprinting process with the feature vector affects the values of the mapped indexes. Therefore, many of the imprints were rendered useless after this type of attack. Despite significant destruction to the image visualisation with an increased rate of modification, it was possible to recapture the feature vector from some of the images, even following enormous alteration, such as when the object was changed by 95%. At the same time, this attack caused significant loss of the mapped index values compared with the experiment that investigated modification in one area, where the latter was vandalised less than the former in terms of the impact on the pixels of interest.

To explain why there is a fluctuation in the resulting graph in both Figure 7.10 and Figure 7.11, the following simulation gives a possible justification of the obtained results of these to attacks extermination. Assume that a vector of [0 1 2 3 4 5 6 7 8 9] needs to be imprinted with this given random matrix of numbers from 0-9 (i.e. pixels or similar type of object) which has dimensions of 7 x 7 (49 values) as shown in Figure 7.12.

4	5	2	1	9	4	9
7	0	1	6	4	3	2
1	9	2	9	2	8	3
7	3	5	8	3	8	8
6	0	2	7	5	0	1
9	8	3	6	1	1	2
9	7	5	0	8	4	0

Figure 7.12. Matrix of numbers to be used as an example of an object to be imprinted with a given vector.

By mapping the given vector value with this matrix, this should result in 4 possible, complete, and non-overlapped imprints scattered among the matrix as shown in Figure 7.13.

4	5	2	1	9	4	8
7	0	1	6	4	3	2
1	9	2	9	0	8	3
7	5	3	8	3	0	1
6	0	2	7	5	0	1
9	6	8	6	1	1	2
4	7	5	0	8	4	0

Figure 7.13. Matches values of the given vector highlighted in colour per imprint

Now, by simulating some attacks (shown in a black background in Figure 7.14, Figure 7.15, Figure 7.16, and Figure 7.17) to reflect why the fluctuation in the results occurred. As shown in Figure 7.14, the yellow, blue and green imprints are affected by the attack in which only the red imprint is still valid to be retrieved. This results in 1 out of 4 possible imprints can be retrieved from this given simulation.

4	5	2	1	9	4	8
7	0	1	6	4	3	2
1	9	2	9	0	8	3
7	5	3	8	3	0	1
6	0	2	7	5	0	1
9	6	8	6	1	1	2
4	7	5	0	8	4	0

Figure 7.14. Biometric information–document correlation generation pipeline

In Figure 7.15, although that the affected area is larger than the previous example (shown in Figure 7.14), only the green imprint is affected by the attack in which the yellow, blue and red imprints are still valid to be retrieved. This resulted in 3 out of 4 possible imprints can be retrieved from this given simulation.

4	5	2	1	9	4	8
7	0	1	6	4	3	2
1	9	2	9	0	8	3
7	5	3	8	3	0	1
6	0	2	7	5	0	1
9	6	8	6	1	1	2
4	7	5	0	8	4	0

Figure 7.15. Biometric information–document correlation generation pipeline

With this attack in Figure 7.16, all imprints are affected by the attack in which no valid imprint to be retrieved. This results in 0 out of 4 possible imprints can be retrieved from this given simulation.

4	5	2	1	9	4	8
7	0	1	6	4	3	2
1	9	2	9	0	8	3
7	5	3	8	3	0	1
6	0	2	7	5	0	1
9	6	8	6	1	1	2
4	7	5	0	8	4	0

Figure 7.16. Biometric information–document correlation generation pipeline

Although the attack shown in Figure 7.17 affected more values of the matrix in comparison with the previous attacks, the yellow imprint is not affected by the attack in which this results in 1 out of 4 possible imprints can be retrieved from this given simulation.

4	5	2	1	9	4	8
7	0	1	6	4	3	2
1	9	2	9	0	8	3
7	5	3	8	3	0	1
6	0	2	7	5	0	1
9	6	8	6	1	1	2
4	7	5	0	8	4	0

Figure 7.17. Biometric information–document correlation generation pipeline

By plotting the number of retrieved imprints in the function of modification rate, it can be seen in Figure 7.18 that the resulted graph is fluctuating in which even with more areas/values modified, the affected imprints can vary. This fluctuating will depends on the object values and the vector values to be imprinted.

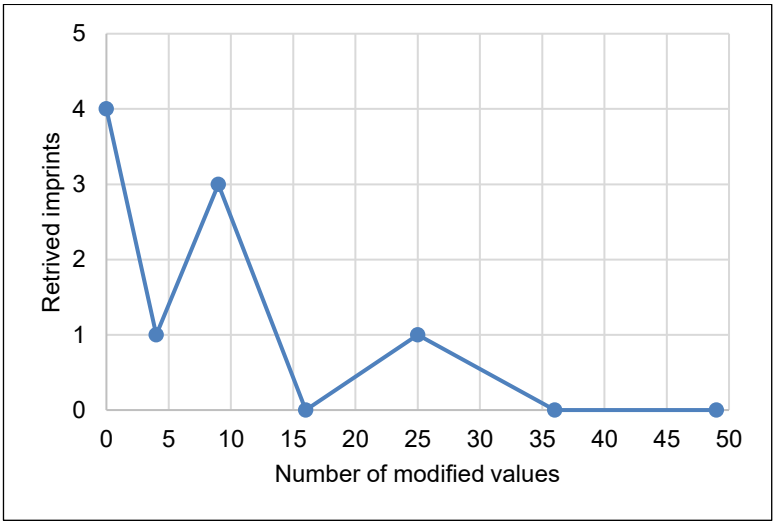

Figure 7.18. Biometric information–document correlation generation pipeline

These simulated attacks in this small matrix/imprints explain why the rate of retrieved imprints have fluctuated at some modification rations and not a steady, smooth decline. In the conducted experimentation, the tested images have

dimensions of 1,234 x 1,858 and 1,858 x 1,234 width, height in pixels, this is equal to 2,292,772 pixels per image which enabled (a minimum of 244 and of maximum 1,815) of imprints to be generated. If only one imprint per image is successfully retrieved, the result is counted valid as we only need one imprint to find a match in the biometric signal.

Finally, in the last experiment, the most striking finding to emerge from the results is that of all the tests in this experiment, the feature vector was retrieved and reassembled 100% for all the images tested. This suggests that by giving only part of the original imprinted image, it is possible to restore the feature vector to its original values. Figure 7.19 shows the percentage of successfully retrieved feature vectors under a partial image attack. The results relating to this attack also show that, as long as the corresponding locations of the imprints are not changed, it is still possible to retrieve the imprinted information, as the mapped indexes are still valid.

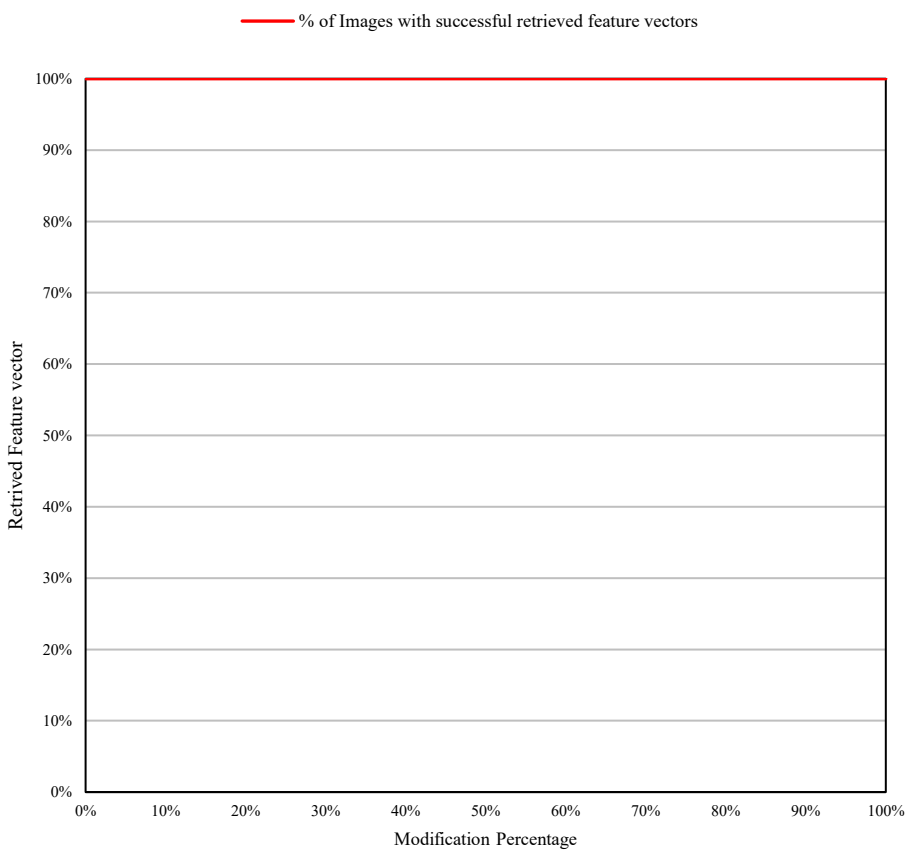

Figure 7.19: Partial image attack

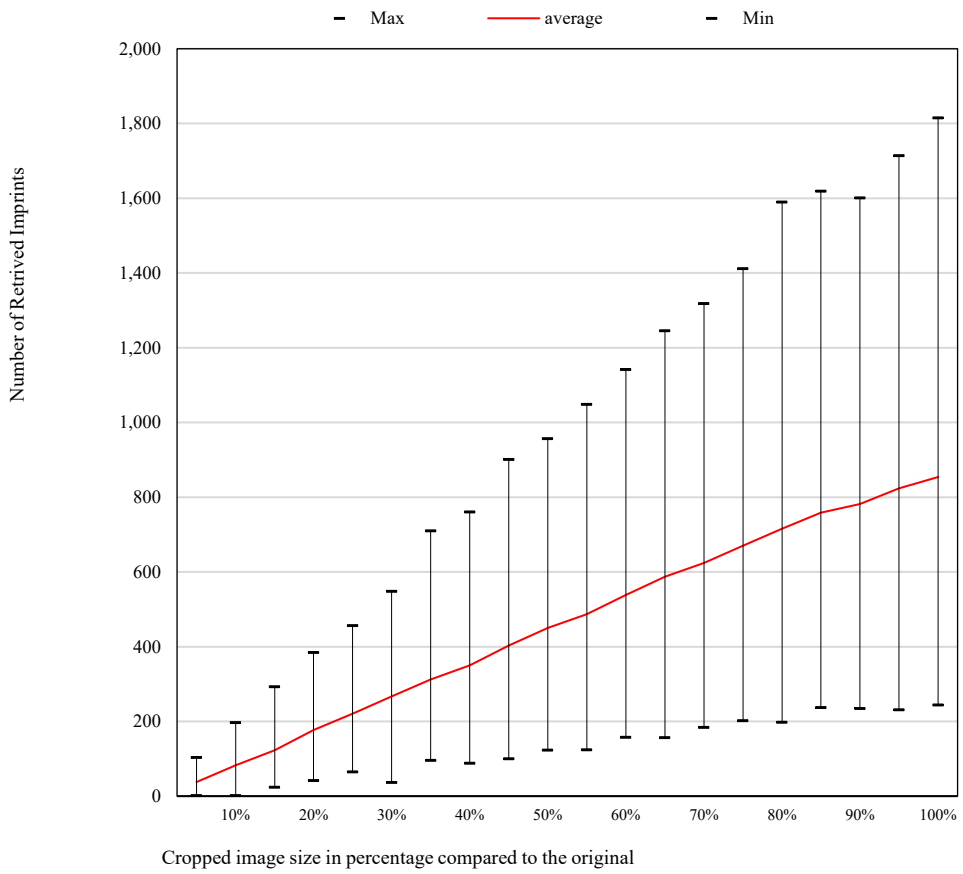

Figure 7.20: Multiple parts modification attack

Figure 7.20 shows that the average, maximum, and minimum numbers of a retrieved feature vector cross for all the images examined. However, these results were obtained by assuming that the preserved indexes of the hexes of interest had not been changed after the cropping process. This suggests that all the imprints in the database correlated with the samples in question as part of the original images. In practice, this is not always possible, since the original object might not be accessible or available after the imprinting process takes place.

Therefore, more research is needed to find a link between the parts of an object and the original.

7.5 Discussion

The results have shown that, by mapping the hex representations of a feature vector to the hex representations of an image of interest, it is feasible that one or more imprints of the feature vector can be generated. The results of the first experiment conducted revealed an 'expected' outcome by imprinting the feature vector from the original imprinted image. Since 100% of the imprinted feature vector was retrieved using only the generated imprints that contained the indexes of the corresponding positions, this was expected because the mapped objects (images in this case) had not been exposed to any kind of alteration and, therefore, were tested based on their original status.

Being able to achieve such high scores is attributable to the nature of the examined object. Since images are a set of pixels that range from 0 to 255, changing the value of one pixel does not affect the value or the position of other pixels. Thus, altering part of an image is not necessary, as it affects the values of all the imprinted indexes. Therefore, generating as many imprints as possible in various positions of the image will, in turn, increase the likelihood of successfully retrieving the imprinted values.

It is worth mentioning that the approach introduced in this chapter could be applied to other types of object, such as Microsoft Office Word documents and PDF files. Nevertheless, the results do not necessarily reflect a robust success rate, since changing a small part these file types could affect the offset of binary

representation of these files. An initial experiment was also carried out in which a small set of Office Word and PDF files were examined using the same imprinting technique that was conducted on the images. The results showed that, unlike images, changing a small value in the document/file content leads to an adjustment of the entire offset/index sequence. For that reason, many attacks would have a considerable impact on the accuracy of retrieving the imprinted feature vector from such objects.

In comparison with the experiment in Chapter 6—Investigation of a Biometric-Based Null Cipher Using Images, the use of a grille cipher increases the likelihood of retrieving the mapped information (biometric signals). Moreover, the grille cipher approach does not change the integrity of the data by directly modifying the examined image, as it only generates those imprints, which is simply a mapping between individual biometric signals and the corresponding matching location within the object. Another advantage of this approach is that it works across different file types, not only images (i.e., the text-based files that are investigated in the next chapter).

Nevertheless, these findings provide interesting insights for future research, whereby other techniques could be investigated for robust object alteration. A possible solution for tackling this type of issue could be that, instead of mapping the feature vector to the object at the hex level, a higher level of representation could be used. For instance, in the case of documents, mapping the feature vector with static representations of the text of the document (such as a locality sensitive hashing digest) might possibly become less vulnerable to an alteration attack,

especially when the imprints generated preserve more static values related to the object. Therefore, the next chapter provides an investigation into leveraging locality sensitive hashing algorithms to generate a feature vector less sensitive to a modification that could be used for establishing the desired document to a user correlation file.

7.6 Conclusion

This chapter proposed a proactive framework that uses transparent biometrics to inextricably link the use of information (images) to the individual users who use and access them, rather than intermediate controls. Such an approach would aid digital forensic investigators in their analysis of electronic evidence and could increase the likelihood of evidence being admissible in a court of law. The results of all the experiments conducted show that even when an object is altered in a sophisticated way, there is still a chance that the imprinted feature vector can be retrieved and reconstructed.

Despite these promising results, the use of transparent biometrics to monitor and acquire a subject's traits introduces several challenges that need to be considered when developing this type of system. For instance, in the case of facial detection, environmental and external factors, such as light, the subject's distance from the camera and facial orientation, significantly affect the accuracy of the samples obtained. Even with extensive research being undertaken in this field, such issues cannot be overcome very easily, primarily because the operation of transparent biometric monitoring is meant to be unobtrusive and unsupervised. Further work is required to investigate the ability to utilise a broader

range of file types, such as text documents (i.e., PDF and DOCX) since different file types operate differently from an embedding perspective.

8 Investigation into a Biometric-Based Grille Cipher Using Documents

The preceding chapter presented promising experimental results in validating the feasibility of the grille cipher approach in generating a forensically rich correlation between the user and the image file is used. This chapter investigates the feasibility of mapping an individual's biometric information to documents by leveraging locality sensitive hashing (LSH) algorithms, such as Trend Micro Locality Sensitive Hashing (TLSH) (Oliver, Cheng and Chen, 2013). The chapter includes an evaluation of the developed approach by measuring the success rate for retrieving the mapped signals to assess how robust the method is given subsequent modification of the document. The chapter concludes with discussion and conclusion sections that examine the findings and highlight the limitations identified.

8.1 Introduction

This investigation sought to provide a technique for mapping between a digital object and biometric identifiers and storing the mapped information alongside document identifiers in a centralised storage repository. When the mapped (imprinted) objects are recovered or analysed, the information stored in the repository is used to recover the biometric information, which is subsequently used to identify the user. The key advantage of this approach is that the underlying digital object is not modified in any way, in contrast with the watermarking techniques referred to previously. No explicit biometric information

is stored, as only the correlation that points to locations within the imprinted object is preserved.

8.2 Methodological Approach

The proposed approach takes advantage of LSH schemes to generate a representation that is less sensitive to modification of a document (text). In general, LSH algorithms are mainly used for dimensionality reduction by mapping high-dimensional input space into a lower-dimensional space. A key difference between LSH-based algorithms and cryptographic schemes is that the former are less sensitive to small changes in the mapped input space. In contrast, hash-based cryptographic schemes are designed for ensuring data integrity by maximising its sensitivity to the input space. Both methods map the input stream into a fixed output called a digest (hash values).

This study leverages the LSH property of maximising the probability of a collision of similar inputs. This is achieved by directly mapping the biometric feature vector representation of an individual with the computed LSH digest of a given document, which generates a digital record—known as an 'imprint' file. The resulting imprint file represents locations within the computed LSH hash value, each of which corresponds to a respective portion of the digital biometric feature vector. The user's biometric samples from which the feature vector is computed (e.g., facial features, iris, keystroke analysis or behavioural profiling) are transparently and continuously captured—using suitable sensors—while the person is interacting with the computer.

Finally, the imprints generated are stored in a centralised, secure database for later analysis when required.

The locality sensitive hashing scheme TLSH is utilised in this research to construct documents digest. The construction of the TLSH digest is computed as the following steps:

1. Process the byte string of the given document using a sliding window of size 5 to populate an array of bucket counts
2. Calculate the quartile points, q1, q2 and q3
3. Construct the digest header values

Construct the digest body by processing the bucket array

Steps 1, 2 and 4 combine to use a modified bit sampling method; instead of bit sampling, these steps are sampling pairs of bits. The sampling process is done to a finite precision so as to have a fixed-length digest. Step 3 constructs innovative features based on the approach used to get a fixed-length digest. The final TLSH digest constructed from the Byte string is the concatenation of the hexadecimal representation of the digest header values from step 3, and the hexadecimal representation of the binary string from step 4 (Oliver, Cheng and Chen, 2013).

Figure 8.1 illustrates the process of generating imprint files, which establishes the correlation between the biometric information acquired from the corresponding person and the triggered document. Data leakage in the form of a document (whether posted on a public website or captured by the network) can then be

analysed by processing the imprint file with the given 'leaked' document, which was already imprinted at some point before it was leaked, to reconstruct the mapped biometric feature vector. Once the sample is extracted, it can be processed by a biometric system in order to determine the last user to interact with the object.

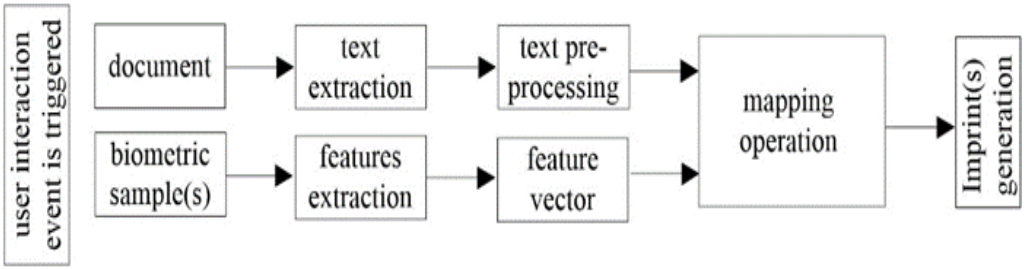

Figure 8.1. Biometric information–document correlation generation pipeline

To illustrate how mapping the biometric feature vector with the LSH digest works, assume that the following feature vector needs to be mapped to the given LSH digest, as shown in Figure 8.2.

	Value
Feature vector sample	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
LSH digest sample	[F1751BD78C133A4A9303D6365E78E4933D843436A7921120789B58138AFB927BF7DE]
Index	0.....10.....20.....30.....40.....50.....60.....

Figure 8.2: Examples of a feature vector and the TLSH digest sample

In this example, each value (digit) of the feature vector exists in more than one location within the hash digest. The sample digest in this figure was computed using the TLSH scheme, the output of which is 70 hex characters long (35 bytes). TLSH is an LSH scheme developed by TrendMicro (Oliver, Cheng and Chen, 2013). In mapping, zero is located in two locations: 18 and 47.

In the same manner, the mapping process finds all matching locations for the remaining values of the given feature vector, as shown in Figure 8.3.

F.V.	Matched index location within the TLSH digest									
0	18	47								
1	1	4	10	44	45	54				
2	43	46	61							
3	11	12	17	19	22	31	32	36	38	55
4	14	29	35	37						
5	3	24	52							
6	21	23	39							
7	2	7	26	41	48	62	65			
8	8	27	34	49	56					
9	16	30	42	50	60					
	1st	2nd								
	imprints									

Figure 8.3: Feature vector–LSH digest mapping matrix

Combining the mapped locations (one location from each row) forms a single imprint. Hence, the total number of unique imprints that can be generated from the mapped indexes is two, as highlighted in light green in Figure 8.3. Therefore, using any of these imprints, it is possible to reconstruct the original (mapped) feature vector from the document by reversing the mapping process. The next subsection describes the correlation-generation pipeline, including the mapping process step.

8.2.1 Correlation-Generation Pipeline

The generation process of the imprint file that associates an individual's biometric signal with a document of interest involves six main steps, starting with acquiring the text from the document and ending with generating the target imprint file.

8.2.2 Extracting document text

The document text is extracted from the file; the text itself is processed, not the document file type. This approach makes it possible to imprint any document type so that its text can be extracted. For example, PDF, DOCX, TXT, HTML or even email messages can all be analysed and their content parsed. Furthermore, the extraction process eliminates any text formatting and, therefore, the subsequent steps of the imprinting process rely purely on the text.

8.2.3 Pre-Processing the Extracted Text

In this phase, all extra spaces between words, lines, paragraphs and pages that exist in the text are removed and replaced with a single space. This ensures that the computed LSH digest is based only on plain text, which means that if the

document is maliciously manipulated later, for instance by adding extra spaces or page breaks, it will have low or even no effect on the computed hash value.

8.2.4 Computing the LSH Value of the Text

The LSH value can be computed by using one of the widely known open-source algorithms, including Ssdeep, Sdhash, Nilsimsa and TLSH (Damiani *et al.*, 2004; Kornblum, 2006; Roussev, 2010; Oliver, Cheng and Chen, 2013). It is well established that TLSH is more robust than the other schemes with regard to digest entropy, collision likelihood, and manipulation attacks (e.g., removing, swapping, and inserting words) (Oliver, Forman and Cheng, 2014). Therefore, the TLSH algorithm was chosen for use in this study to compute the hash digest of the extracted text. Two approaches can be used to compute the hash digest of the document:

a) Only a single hash digest is computed for the whole document, which makes the imprinting process much faster and stores fewer data in the database, as only one digest is used to generate the correlation with the biometric signal.

Hashing the text using a different resolution to produce multiple digests per document, for example, per page, half page, and paragraph, or using the k-overlapped-folds of the examined document, as illustrated in

b) Figure 8.4. The figure presents how the document text is sliced into 10-overlapped-folds, each of which is processed separately and its LSH value computed.

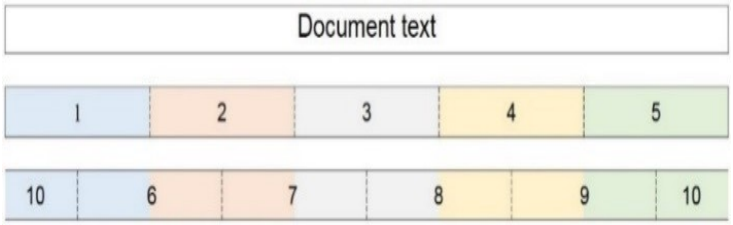

Figure 8.4: Slicing document text into 10-overlapped-folds

In this study, methods (a) and (b) are both examined and evaluated against different possible attack vectors, as detailed in section 8.3.

Another LSH hash digest is computed (using, for instance, Nilsimsa) and stored in a centralised database to be used later to locate the associated imprint file when a document is queried. The biometric signal is also hashed using the Secure Hash Algorithm (SHA) digest and stored. The SHA is used for checking the integrity of the extracted biometric signal. The reason for using another LSH algorithm is to avoid storing the same LSH digest that was used for generating the imprint. This ensures that having only the imprint in the database without the correlated document makes it impossible to reconstruct the related biometric information.

8.2.5 Mapping the Feature Vector to the Hash Digest Value

The feature vector and the LSH hash value of the text are mapped to its equivalent location in the text LSH hash value to retrieve the locations where they might match, as described previously in this section.

8.2.6 Generating Imprints

By retrieving the locations of each character in the feature vector with the object, it becomes possible to generate imprints based on the list of indexes obtained. This means that multiple imprints of the whole feature vector can be generated by combining those positions.

8.2.7 Recovery Algorithm

The recovery algorithm used to extract and reconstruct the imprinted biometric information from a questioned document in the case of information leakage shares the same steps 1-3 of the imprinting process listed above. This is followed by the steps given below:

1. The questioned document hash digest is computed (e.g., Nilsimsa) as input to the next step.
2. The related stored imprint file is retrieved by querying the centralised database--using the computed hash digest--where previously generated fingerprints and imprints for all documents are stored.
3. The retrieved imprint file is mapped to the computed LSH value of the document in question, and the correlated biometric signal is reconstructed from the mapped locations.
4. To validate the integrity of the reconstructed biometric signal, its SHA digest is compared against the stored digest generated when the imprint was created.

After explaining how the imprinting and retrieval techniques of the proposed approach work, the next section investigates the feasibility of imprinting biometric

information with documents and later recovering them (even after the text has been modified).

8.3 Experimental Analysis

The fundamental research question concerning the imprinting of a biometric signature is how robust the approach is given subsequent modification of the document–arguably, the key attack vector against this approach. An insider who intends to leak a confidential document could maliciously manipulate its content in order to destroy any tracks to avoid being traced. Therefore, to examine the feasibility and effectiveness of the proposed approach, genuine leaked documents from WikiLeaks were chosen for experimental purposes. WikiLeaks is an international non-profit organisation that publishes secret information, news leaks and classified media provided by anonymous sources (*WikiLeaks*, no date). In 2009, it released more than 6,000 reports commissioned by the United States Congress. These reports are classified as confidential documents but are now publicly available and accessible online in the form of text files (WikiLeaks.org, 2009)..

Table 8.1 provides statistical information about the dataset used. Leaking repositories, such as WikiLeaks and The Intercept, typically perform some kind of modification to the leaked documents. For instance, they watermark uploaded documents and files with extra information, such as a document ID, date, website address or logo (*The Intercept*, no date).

Table 8.1. Corpus statistics

File size distribution (KB)	# of docs	Document content	Min.	Max.	Average
1-99	4,920	Chars.	1,288	874,548	47,345
100-199	853	Words	233	155,614	8,873
200+	227	Lines	38	16,160	981
Total	6,000	Pages	1	622	34

A number of experiments were designed and conducted to evaluate the proposed approach in scenarios that consider malicious intent with regard to any possible modification that might be performed on a document. The first experiment maps the biometric feature vector to the computed text TLSH digest and retrieves it. The goal was to compute the possible number of imprints that could be generated from the mapping process. A total of 21 attacks were developed and included file, formatting and text-based manipulation methods. The attacks were employed to examine critically the effectiveness of possible modification attacks on the imprinted documents and how such attacks could affect the retrieval performance of the mapped biometric information. The developed attacks can be classified into three main categories: file-type conversion, formatting change and content manipulations, as shown in Table 8.2.

Table 8.2. Possible document manipulation methods

File-type conversion	Formatting change	Content manipulation
1.PDF to .docx	9. Font resizing	14. Deleting words
2.PDF to .txt	10. Font type changing	15. Deleting sentences
3.PDF to Image	11. Colour changing	16. Deleting lines
4.Docx to PDF	12. Text highlighting	17. Swapping words
5.Docx to txt	13. Line and para spaces	18. Swapping sentences
6.Txt to PDF		19. Swapping lines
7.Txt to .docx		20. Substituting synonyms
8.Txt to Image		21. Inserting new words

In the imprinting process, the biometric feature vectors used to represent real facial features. The Fisherfaces feature extraction algorithm was used to compute these vectors for the captured users' facial images (Belhumeur, Hespanha and Kriegman, 1997). The dimensions of the generated feature vector when using the Fisherface algorithm are small compared with deep learning approaches, as the length of the vector is a prime factor when performing the imprinting process. The resulting vector is four-dimensional, with a length of 60 digits. The chosen vector includes the frequency of all digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), as well as a '-' sign, to ensure that this study covers all possible numbers within the mapping process.

For all the manipulation methods referred to above, the original document TLSH value was computed before it was modified and the resulting digest was then imprinted with the biometric information. After that, the manipulation methods were applied to the imprinted documents. Finally, the TLSH value of the modified

version was computed again and compared with the original. As long as the original text has not changed, the fully mapped biometric feature vector should be successfully retrieved by reversing the imprinting process. However, this is not always the case, since a leaked document is highly likely to have been manipulated or modified. Consequently, the computed hash value is directly affected, to what degree depends upon the scale of the modification. Fortunately, the TLSH is less sensitive to small changes than cryptographic hashing algorithms, such as the SHA, since a small modification in the input drastically changes the output computed digest. This is the so-called avalanche effect.

In contrast, all similar digest schemes have the property that a small change to the file being hashed results in a small change to the hash (Oliver, Cheng and Chen, 2013). Figure 8.5 shows two samples of computed document hash digests using SHA256 and TLSH. Each present two values: one for the original document and one for the modified version of the same document.

It is clear in the above that the digest of the modified document computed by SHA256 is entirely different from the originals. The TLSH digest is only slightly affected; the characters shown in red are those that have changed, while the others remain the same with exact locations. Therefore, the TLSH can be used in this approach to give a less sensitive representation of the whole document.

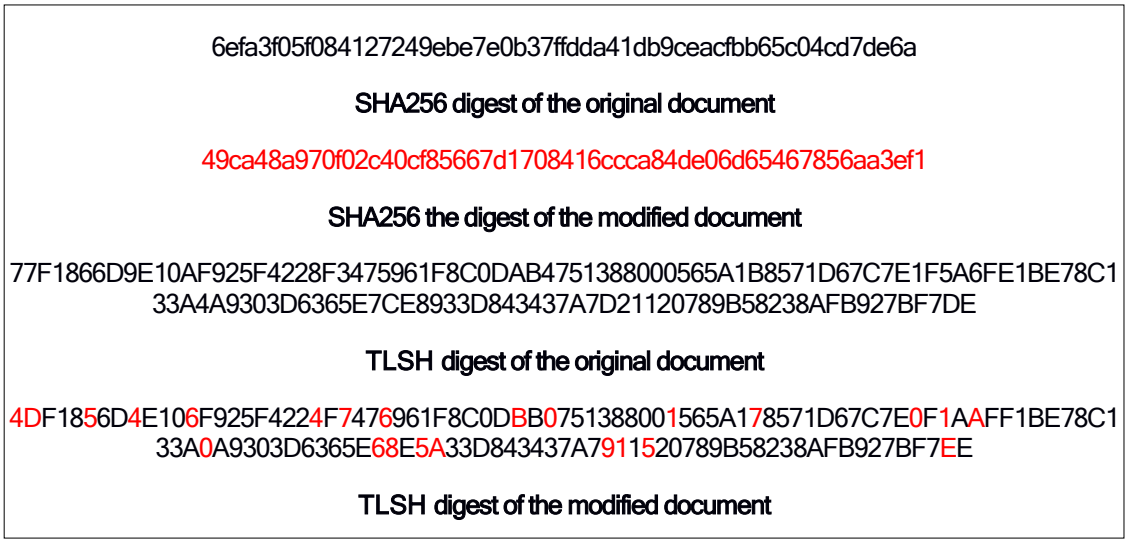

Figure 8.5: Samples of computed document hash digests using SHA256 and TLSH

Figure 8.6 shows the average distribution of the number of imprints generated per document for the 6,000 documents examined in the dataset. The histogram indicates that most of the imprinted documents generated more than three imprints. The number of obtainable imprints mainly depends on the TLSH digest entropy generated and the digit frequency. The rate of entropy and the frequency differ from one document to another, as this is a natural property of hash schemes. Although multiple imprints per document were generated, as the figure illustrates, only one imprint was needed to reconstruct the biometric information successfully. Indeed, having multiple imprints for a given document significantly increases the likelihood of recovering the mapped information, even after the document is exposed to manipulation.

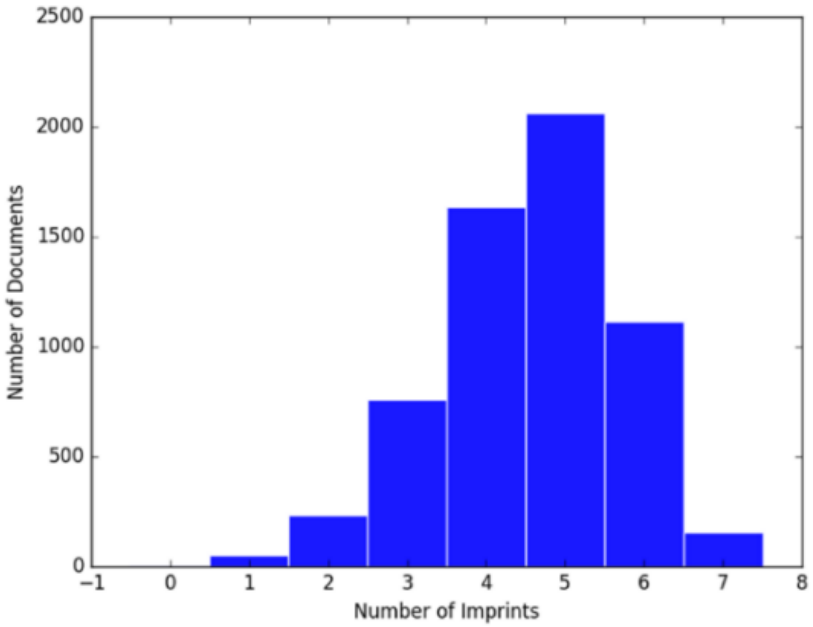

Figure 8.6: Distribution of the imprints generated per document

The experimental results of the method developed indicate that the proposed approach is resistant and robust against both file-type conversion and formatting change attacks, with an accuracy of 100%. Since the nature of these modification methods does not change the actual text or content that is fed into the LSH algorithm, the mapped biometric signal is fully retrievable, even when the text format or file type is changed, including converting the document into an image. However, in such a case, optical character recognition (OCR) technologies could be used to analyse and convert the image content (printed text) into machine-encoded text. In this study, test documents were converted into images (JPEG) to simulate such an attack, and a Tesseract-OCR engine was used to read all the images and recognise and extract the embedded text (Smith, 2007). As long as the OCR was able to recognise the correct text, which it did, the integrity of the text can be preserved when compared with its original version. For the content

manipulation attacks, random settings were configured for the rate of modification, as Figure 8.7 illustrates, ranging from 1 to 100 for word-type attacks and from 1 to 20 for line and paragraph attacks. As this rate increases, the number of changes also rises. For instance, in the case of a word-deleting attack, a number of random words (between 1 and 100) are deleted from each document in the dataset. This also applies to all other attacks that fit the same category.

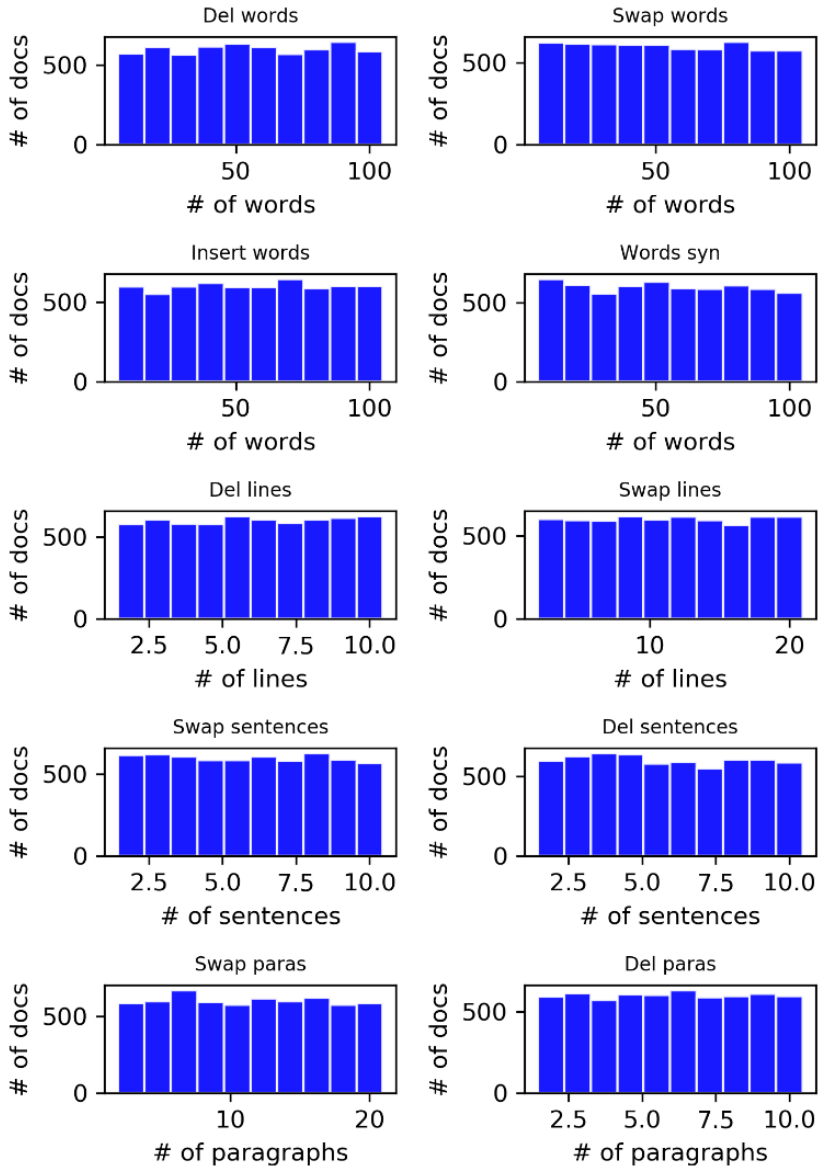

Figure 8.7: Modification types and rates among the dataset documents

Table 8.3 presents the results of retrieving the mapped feature vector under content manipulation attack methods.

Table 8.3. Content manipulation attack methods experimental results

No.	Attack type	Rate (number)	# of retrieved F.V.	Score (%)	TLSH diff. (original/modified) [1]		
					Min.	Max.	Avg.
1	**Del words**	1-100	5,359	89.31	0	217	8
2	**Swap words**	1-100	5,464	91.06	0	82	7
3	**Insert words**	1-100	5,304	88.40	1	471	33
4	**Words syn.**	1-100	5,751	95.85	1	465	30
5	**Del lines**	1-10	2,708	45.13	7	466	43
6	**Swap lines**	1-10	2,637	43.95	7	874	71
7	**Swap sentences**	1-10	5,929	98.81	0	30	3
8	**Swap paras**	1-10	2,853	47.55	5	125	26
9	**Del paras**	1-10	2,767	46.11	5	149	26
10	**Del sentences**	1-10	4,915	82.00	1	788	15
11	**Multi attacks** [2]	1-10	3,828	64.00	1	456	31

[1] TLSH diff. is the distance score between two digests (texts).

[2] A number of attack methods were randomly chosen.

In addition, the TLSH uses a distance score of zero, which indicates that the files are identical (or nearly identical), while scores above that represent a greater distance between the examined documents. A higher score should indicate that there are more differences between the documents (Oliver, Cheng and Chen, 2013). From the data in Table 8.3, it can be seen that given the ability to recover biometric identifiers under significant levels of modification—such as deleting 100

words—it is still possible to regenerate the correlation established between the biometric information and the imprinted document with a success rate of 89.31%. In addition, Figure 8.8 illustrates how accuracy changes along with a defined number of deleted words.

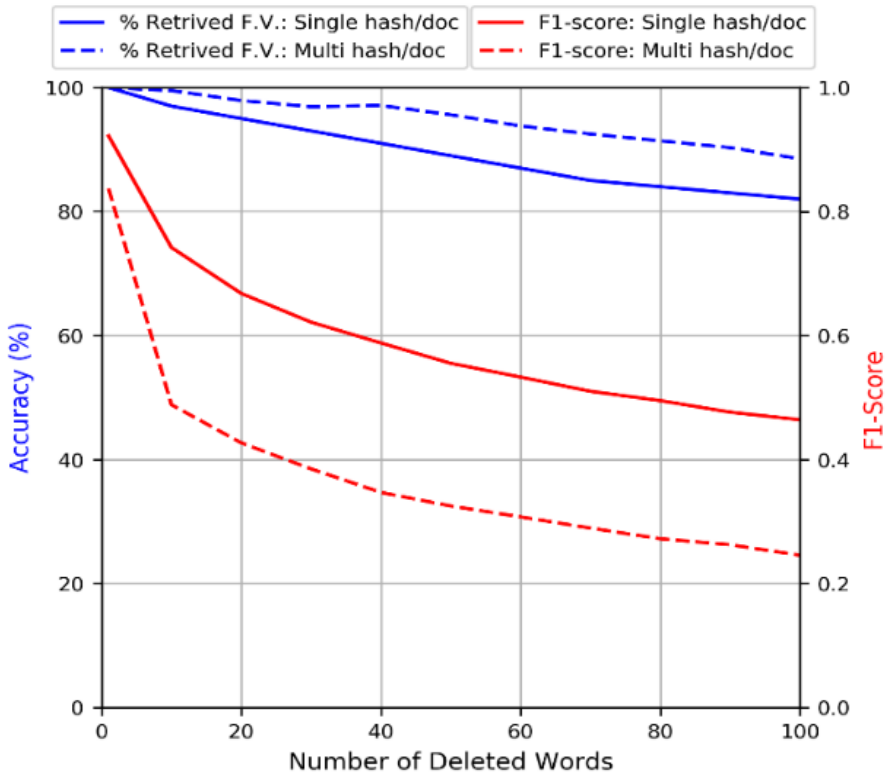

Figure 8.8: Averaged accuracy and F1-Score for a deleted words attack

Two levels of hashing resolution were applied on the examined documents: one hash digest per document and a multi-hash digest using 10-overlapped-folds per document. The overall accuracy improved when multi-hash digests were generated. In general, a document was counted as correctly identified (feature vector retrieved) if at least one imprint was perfectly extracted from the imprinted

feature vector, even when the computed hash digest was not identical to the one from which the original correlation was established.

Furthermore, paragraph attacks (swap and delete methods) scored low rates, of 47.55% and 46.11%, respectively. These rates are considered low as biometric information of more than half of the documents could not be reconstructed due to the effect of the attack. Indeed, removing a number of paragraphs from the document significantly affected the computed hash digest to a greater degree than other types of modification, such as deleting words or sentences. This can be improved by changing the hashing resolution (i.e., using k-folds). For instance, instead of hashing the whole document and generating a single hash digest, multiple digests were computed for the document, for example per page, half page or paragraph, and correlated the biometric information with the resulted hashes.

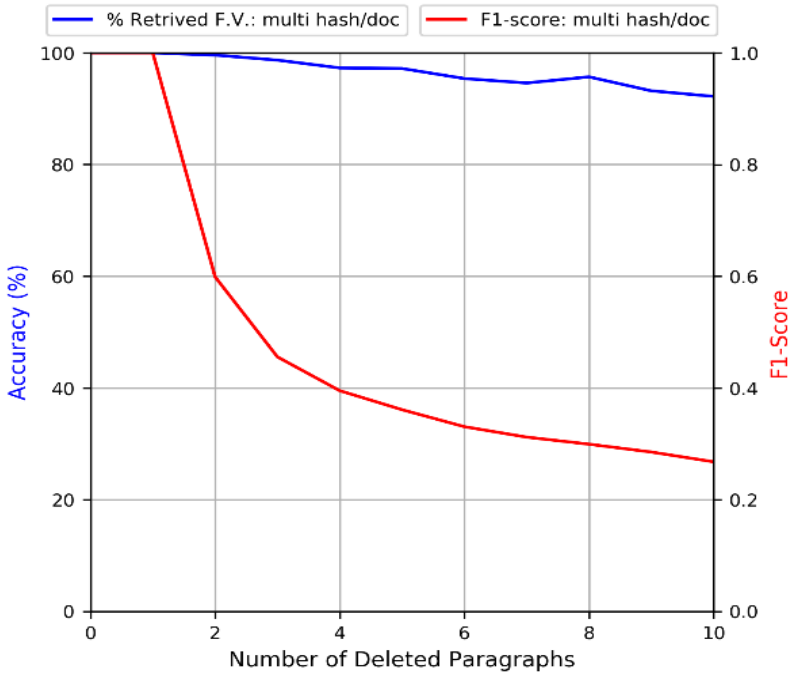

Figure 8.9: Averaged accuracy and F1-Score for a deleted paragraphs attack

Figure 8.9 shows the averaged accuracy and F1-Score for a deleted paragraphs attack using 10-overlapped-folds. The overall accuracy is higher than for the single hash digest per document approach, as it scored 93%. In contrast, the F-Score achieved is not high, as it was computed for all the generated imprints, while only one valid retrieved imprint of a given document was needed to reconstruct the mapped biometric information. Moreover, the chances of recovering correlated biometric signals vary based on the type and scale of the attack vector. However, in many leakage cases, the leaked document might not be exposed to severe modification. Hence, reconstructing the biometric sample is highly likely to be possible and, as a result, the source of the leak can be identified.

8.4 Discussion

The most apparent finding to emerge from this study is that the underlying digital objects, documents, in this case, are not modified in any way. The proposed approach also disassociates any biometric information from the digital object itself, thereby minimising any attacks on the biometric data. This suggests that the biometric signal is not stored by any means in a database; it is only its correlation to the imprinted object (document/text in this case) that is preserved in the imprint file. Thus, the signal becomes useless without the presence of the imprinted document in the recovery process, since the imprint file that correlates the object with the related biometric signal only contains those locations within the document from which the signal can be extracted. It also allows larger volumes of information to be imprinted, making it more suitable for digital objects when

more significant levels of information need to be correlated (i.e., multimodal biometric samples). It does, however, introduce the need for a centralised repository which will grow as users interact with objects and thus require configuration and management.

Although the above investigation has critically examined the proposed approach against possible malicious attacks and shown robustness and strength, some challenges exist and require further research. These include the ability to automate the process of capturing biometric signals and detecting user interaction with the object instantly, along with establishing the correlation with the interacted object. This requires the development of a smart and active agent that continually captures an individual's biometric information (using a camera in the case of facial information) and performs the imprinting process.

Furthermore, the proposed approach raises substantial privacy concerns for those individuals who are monitored by the system, as processing, transmitting and storing biometric samples into a centralised database requires a high level of confidentiality and sufficient resources. This obviously needs to be investigated in-depth in future work. More broadly, research is also needed to determine the ability to utilise a broader range of digital objects. Different objects have varying degrees of stability due to their structure. For example, executable files and their underlying data structure can change considerably given small alterations to a file, in contrast with the text. Therefore, the proposed approach needs to be examined for such file types to measure its usefulness and robustness fully. Further study also needs to be carried out regarding the ability to utilise soft

biometric features, such as the gender, age and even race of individuals, to increase the discriminative ability and provide more reliable information to the investigator.

8.5 Conclusion

This chapter introduced a proactive approach to aiding an incident investigator to establish and examine a case of insider misuse, particularly with respect to information leakage, which could increase the likelihood of the evidence being admissible in a court of law. This study has shown that it is possible to recover biometric information from text files successfully, even under significant modification attacks. Rather than requiring the complete digital object, it is possible to recover the necessary information with even a modified version of the questioned document. This was achieved by utilising the TLSH algorithm to enable the creation of a compact representation of the examined text files, in which mapping these with biometric feature vectors becomes less sensitive to malicious modification. Based on the experimental results obtained, when the imprinted documents are significantly modified (i.e., deleting more than two paragraphs of a document that has 20 paragraphs), the imprinted values can be lost. At which point, retrieving and reconstructing mapped biometric information becomes less likely to be successful. Therefore, further investigation and experimentation into leveraging different locality sensitive hashing or dimensionality reduction schemes and algorithms are strongly recommended. Some possible future studies using the same experimental setup are apparent. It would be interesting to assess the effects of employing deep learning algorithms,

such as autoencoders, to combine the original object with the biometric information. As such, this could generate a robust correlation that is resistant to more sophisticated modifications.

9 Conclusions and Future Work

This chapter concludes the thesis by outlining the key contributions and achievements of the research. This is followed by a summary of the limitations and obstacles encountered during the development of the research project. Finally, potential areas of further research work are also presented.

9.1 Contributions and Achievements of the Research

The research has accomplished all the objectives stated initially in chapter 1, with a series of experimental studies leading to the development of a proactive biometric-enabled forensic imprinting system. The key contributions and achievements of this research are as follows:

- Proposing a novel proactive digital forensic approach that correlates individuals with the digital objects with which they interact. The proposed approach operates in two modes—centralised or decentralised—each of which overcomes the limitations of the other. For instance, the centralised mode can map biometric signals with multiple file types, while the decentralised mode requires a specific embedding technique for each file type. However, the latter does not require a centralised database in order to operate.
- Conducting an investigation into biometric-based image steganography by embedding individuals' biometric signals directly into image files, with a particular focus upon the ability to recover the biometric information under varying degrees of modification attack. The experimental results show that

even when a watermarked object is significantly modified (e.g., only 25% of the image is available), it is still possible to recover the embedded biometric information.

- Conducting an investigation into biometric-based image imprinting by linking a subject (i.e., a computer user) with an object of interest (e.g., images) using the individual's biometric sample, such as a facial biometric, without modifying the object being imprinted. This investigation has also developed a set of experiments that employ a grille cipher technique to generate a correlation that could identify the individual. The experimental results of the proposed approach have shown that it is possible to correlate an individual's biometric feature vector with images and still successfully recover the biometric information, even with significant file modification.
- Conducting an investigation into biometric-based document mapping to attribute document misuse via information leakage using biometrics and a locality sensitive hashing scheme. Comprehensive experiments using the proposed approach have shown that it is highly possible to establish this correlation even when the original version has undergone significant file modification. In many scenarios, such as changing the file format or removing parts of the document, including words and sentences, it was possible to extract and reconstruct the correlated biometric information from a modified document (e.g., 100 words were deleted) with an average success rate of 89.31%.

A number of papers related to the research programme have been presented and published in refereed journals and conferences. As a result, the research is

deemed to have made positive contributions to the field of proactive digital forensics, and specifically in the biometric identity tracing domain.

9.2 Limitations of the Research

The proposed proactive biometric-based forensic imprint system provides a proactive digital forensic approach to inextricably link the use of information (e.g., documents and images) to the individual users who use and access them, through the use of steganography and transparent biometrics. Whilst the set of experimental tests conducted with the system has the foundation for providing such a link, several limitations and concerns need to be considered and addressed before ubiquitous adoption and fully effective operation of the system. The key limitations of this study are briefly listed below.

- In the investigation into an experiment on a biometric-based null cipher using images, the placement of the bits to be embedded (payload) could be randomised using a random function, so that the payload is scattered among the image's pixel space. It is then almost impossible to retrieve the embedded data without knowing the seed of the random function. Although this is an instrumental approach, the recovery of the embedded payload (biometric information) is only possible if the image has not been modified or cropped after the data were embedded.
- Using the LSB for inserting the payload is vulnerable to file-type transformation attacks. For instance, transforming the resulting image (e.g., PNG) to other formats (e.g., JPEG) is highly likely to lead to the destruction of the embedded data by the compression algorithm. Therefore, a more

robust null-ciphered approach could be used, in which, instead of using a conventional spatial technique (LSB substitution) that embeds the data directly in the intensity of the pixels, other approaches, such as frequency techniques, can be used, in which images are first transformed into an intermediate image and then the data are embedded in the image.

- An approach that involves documents with a grille cipher raises significant privacy concerns for those individuals who are monitored by the system. Processing, transmitting and storing biometric samples into a centralised database require a high level of confidentiality and sufficient resources. It is clear that this needs to be investigated in depth in future work.
- The proposed proactive digital forensic approach relies mainly on an individual's biometric information being captured transparently. Hence, the robustness and quality of the captured samples vary dramatically, depending on the stability and the conditions in which the samples were captured. Therefore, further investigation is needed to examine the robustness and reliability of transparent biometric samples in relation to the identification of the individual to whom the captured sample belongs.
- The proposed approach was examined in generating a correlation of the last person who modifies or accesses a digital object, while in more realistic scenarios, the object (such as a document) could be modified or leaked by more than one person. Tracing the leak back to the source is more challenging, as the timing factor needs to be further investigated and the research focused only on generating a correlation and the ability to retrieve it.

- The experiments conducted were all performed offline, while a more realistic study could evaluate the proposed approach in real-life settings while the computer user is performing normal daily tasks. Hence, not only would the retrieval of the biometric samples be measured as a performance metric, the other operations could also be considered, such as the operational overhead of the proposed system with regard to CPU and memory consumption.
- Whilst the system is designed to be able to identify information leakage and the misuse of classified/secret data, yet there is one type of category which is whistleblowers that means it will be possible to identify whistleblowers. A whistleblower is a person who exposes secretive information or activity that is deemed illegal, unethical, or not correct within a private or public organisation. Thus, future work should consider addressing the issue of identifying whistleblowers by the system. In which the system should be compliance with the organisation's whistleblowing policies. For example, what is the procedure to be undertaken when the system identifies a whistleblowing incident? Will be there any protection for the Whistleblower such as using the United Kingdome criminal law to protect whistleblowers?

9.3 Ethics and the Moral Context and Implications of the Research

As the proposed proactive biometric-based forensic imprint system inextricably link the use of information (evidence) to the individual users who access and use it by utilising individual biometrics, yet the system has some possible ethical

implications. The relevance of ethical implications of the developed system includes those issues raised by processing, storing and managing individuals' biometrics. Thus, individual rights such as the protection of personal data, confidentiality, and personal liberty, the relationship between individual and collective rights. Biometrics is one of the most significant examples of how complex it is to match individual and collective needs. It indeed leads to questions associated with an individual, social and collective identity which according to some (Mordini and Petrini, 2007). As the presented system in this research is designed to capture, process, store information, containing user-specific data such as digital objects usage data and other personal biometric features to a centrally managed system. This introduces some privacy issues, which typically would not be tolerable for a real-world implementation. Therefore, despite specific measures have been suggested in the system design (e.g. encrypting the biometric feature vector), further practical consideration should be addressed in any future enhancement. Moreover, more work is needed on securing the storage of biometrics-based information to reduce the privacy concern about centralising such information.

In addition, leveraging facial recognition in this research to generate biometric feature vectors which is used as the main lead to match individuals, actions, along with the digital object that they could interact with, have some ethics areas include necessity, complicity, impartiality, bias, and accountability. Concerns surround the issue of privacy, data control, the right that users to remove their own data from the centralised system(s). Also, the implication of having a biased facial recognition algorithm could lead to a false match/result in which the system could

accuse an innocent person. Therefore, thoroughly examining the used algorithms against any potential biases is essential to ensure that the system does not compromise any ethical constraints.

Finally, as the proposed system acts as an active monitoring engine, in the case of using an endpoint camera (e.g. a fitted web camera) this could lead to capture unnecessary user-related activities, such calling on the phone or talking to a friend without interaction with the computer. Therefore, monitoring and tracking processes should be limited to those activities directly related to digital objects.

9.4 Suggestions and Scope for Future Work

This research project has advanced the field of proactive digital forensics in general and insider misuse identification in particular. However, a number of areas for future work exist that are specifically related to this research. These suggestions are detailed below.

- More research is needed to determine the ability to utilise a broader range of digital objects. Differing objects have varying degrees of stability due to their structure. For example, executable files and their underlying data structure can change considerably given small alterations to a file, in contrast with the text. Therefore, the proposed approach needs to be examined for different file types to measure fully its usefulness and robustness.

- Further study also needs to be carried out regarding the ability to utilise soft biometric features, such as the gender, age and even race of individuals, to offer discriminative ability and information to the investigator.
- Further research could also consider the ability to automate the process of capturing biometric signals and detecting user interaction with the object instantly, along with establishing the correlation with the interacted object. This requires the development of a smart and active agent that would continually capture an individual's biometric information (using a camera in the case of facial information) and perform the imprinting process as the capture is taking place.
- With the recent advances in deep learning algorithms, more advanced facial feature extraction techniques can be explored, including utilising deep learning algorithms to generate more robust and less noisy feature representations of a person's facial image.
- Further investigation and experimentation into the liveness of biometric samples are strongly recommended. It would be interesting to assess the effects of using fake biometric samples, such as printed pictures, and to develop a counter machine learning algorithm that could detect such a forgery. This would reduce the chance of cheating in front of the camera so that the system would be able to detect an attack vector.

References

Al Abdulwahid, A. *et al.* (2016) 'Continuous and transparent multimodal authentication: reviewing the state of the art', *Cluster Computing*, 19(1), pp. 455–474. doi: 10.1007/s10586-015-0510-4.

Abed, L. *et al.* (2019) 'Securing Cloud Storage by Transparent Biometric Cryptography', in, pp. 97–108. doi: 10.1007/978-3-030-12942-2_9.

Al-Bayati, B. *et al.* (2018) 'Misuse Detection in a Simulated IaaS Environment', in *International Workshop on Emerging Technologies for Authorization and Authentication*, pp. 103–115. doi: 10.1007/978-3-030-04372-8_9.

Al-kawaz, H. *et al.* (2018) 'Advanced Facial Recognition for Digital Forensics', in *17th European Conference on Cyber Warfare and Security*. Oslo, Norway.

Alharbi, S., Weber-Jahnke, J. and Traore, I. (2011) 'The proactive and reactive digital forensics investigation process: A systematic literature review', *International Journal of Security and its Applications*, 5(4), pp. 59–72.

Almehmadi, A. and El-Khatib, K. (2014) 'On the Possibility of Insider Threat Detection Using Physiological Signal Monitoring', in *Proceedings of the 7th International Conference on Security of Information and Networks - SIN '14*. New York, New York, USA: ACM Press, pp. 223–230. doi: 10.1145/2659651.2659654.

Alneyadi, S., Sithirasenan, E. and Muthukkumarasamy, V. (2014) 'A Semantics-

Aware Classification Approach for Data Leakage Prevention', in *Information Security and Privacy*, pp. 413–421. doi: 10.1007/978-3-319-08344-5_27.

Alruban, A. *et al.* (2016) 'Proactive Biometric-Enabled Forensic Imprinting', in *The International Conference On Cyber Incident Response, Coordination, Containment & Control (Cyber Incident 2016)*. London, UK.

Alruban, A. *et al.* (2017) 'Insider Misuse Attribution using Biometrics', in *Proceedings of the 12th International Conference on Availability, Reliability and Security - ARES '17*. New York, New York, USA: ACM Press, pp. 1–7. doi: 10.1145/3098954.3103160.

AmazonRekognition (2015) *Amazon Rekognition*. Available at: https://aws.amazon.com/rekognition/ (Accessed: 16 March 2019).

AmXecure (2013) *PrivacyID*. Available at: http://www.amxecure.com/index.php/zh/siem/453-privacyid (Accessed: 6 March 2019).

Apple Inc. (2016) *A Message to Our Customers*. Available at: http://www.apple.com/customer-letter/ (Accessed: 15 March 2016).

Ashbourn, J. (2015) *Practical Biometrics: From Aspiration to Implementation*. 2nd edn. Springer. Available at: http://searchsecurity.techtarget.com/tip/Practical-biometrics.

Balinsky, H., Perez, D. S. and Simske, S. J. (2011) 'System Call Interception Framework for Data Leak Prevention', in *2011 IEEE 15th International*

Enterprise Distributed Object Computing Conference. IEEE, pp. 139–148. doi: 10.1109/EDOC.2011.19.

Baltrusaitis, T., Robinson, P. and Morency, L.-P. (2016) 'OpenFace: An open source facial behavior analysis toolkit', in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, pp. 1–10. doi: 10.1109/WACV.2016.7477553.

Belhumeur, P. N., Hespanha, J. P. and Kriegman, D. J. (1997) 'Eigenfaces vs. fisherfaces: Recognition using class specific linear projection', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7), pp. 711–720. doi: 10.1109/34.598228.

Birk, D. and Wegener, C. (2011) 'Technical Issues of Forensic Investigations in Cloud Computing Environments', in *2011 Sixth IEEE International Workshop on Systematic Approaches to Digital Forensic Engineering*. IEEE, pp. 1–10. doi: 10.1109/SADFE.2011.17.

Bishop, M. and Gates, C. (2008) 'Defining the insider threat', in *Proceedings of the 4th annual workshop on Cyber security and informaiton intelligence research developing strategies to meet the cyber security and information intelligence challenges ahead - CSIIRW '08*. New York, New York, USA: ACM Press, p. 1. doi: 10.1145/1413140.1413158.

Bouslimi, D. and Coatrieux, G. (2016) 'A crypto-watermarking system for ensuring reliability control and traceability of medical images', *Signal Processing: Image Communication*. Elsevier, 47, pp. 160–169. doi:

10.1016/j.image.2016.05.021.

Bowles, S. and Hernandez-Castro, J. (2015) 'The first 10 years of the Trojan Horse defence', *Computer Fraud & Security*. Elsevier Ltd, 2015(1), pp. 5–13. doi: 10.1016/S1361-3723(15)70005-9.

Brenner, S. W., Carrier, B. and Henninger, J. (2004) 'The Trojan Horse Defense in Cybercrime Cases', *Santa Clara Computer & High Technology Law Journal*, 21(1), pp. 1–53.

Brown, C. S. D. (2015) 'Investigating and prosecuting cyber crime: Forensic dependencies and barriers to justice', *International Journal of Cyber Criminology*, 9(1), pp. 55–119. doi: 10.5281/zenodo.22387.

Budgen, D. and Brereton, P. (2006) 'Performing systematic literature reviews in software engineering', in *Proceeding of the 28th international conference on Software engineering - ICSE '06*. New York, New York, USA: ACM Press, p. 1051. doi: 10.1145/1134285.1134500.

Cao, Y. *et al.* (2018) 'Coverless Information Hiding Based on the Molecular Structure Images of Materia', *Computers, Materials & Continua*, 54(2). doi: 10.3970/cmc.2018.054.197.

Cappelli, D., Moore, A. and Trzeciak, R. (2012) *The CERT Guide to Insider Threats: How to Prevent, Detect, and Respond to Information Technology Crimes (Theft, Sabotage, Fraud)*. Available at: http://books.google.com/books?hl=en&lr=&id=Rl_44PuReBkC&oi=fnd&pg=PR7&dq=The+CERT+Guide+to+Insider+Threats+How+to+Prevent,+Detect+and+R

espond+to+Information+Technology+Crimes+(Theft,+Sabotage,+Fraud)&ots=ZIrxDGOh97&sig=OhbOggntxIuh_VksY_YJ0fNPS7o.

Carbone, F. (2014) *Computer Forensics with FTK*. 1st edn. Birmingham: Packt Publishing Ltd.

Carrier, B. (2003) 'Defining Digital Forensic Examination and Analysis Tools Using Abstraction Layers', *International Journal of Digital Evidence*, 1(4).

Casey, E. and Stellatos, G. J. (2008) 'The impact of full disk encryption on digital forensics', *ACM SIGOPS Operating Systems Review*, 42(3), p. 93. doi: 10.1145/1368506.1368519.

Ceccarelli, A. *et al.* (2015) 'Continuous and Transparent User Identity Verification for Secure Internet Services', *IEEE Transactions on Dependable and Secure Computing*, 12(3), pp. 270–283. doi: 10.1109/TDSC.2013.2297709.

Chaabane, F., Charfeddine, M. and Ben Amar, C. (2013) 'A survey on digital tracing traitors schemes', in *2013 9th International Conference on Information Assurance and Security (IAS)*. IEEE, pp. 85–90. doi: 10.1109/ISIAS.2013.6947738.

Chan, C.-K. and Cheng, L. M. (2004) 'Hiding data in images by simple LSB substitution', *Pattern Recognition*, 37(3), pp. 469–474. doi: 10.1016/j.patcog.2003.08.007.

Charbonneau, S. R. D. J. and Simon, E. J. (2014) 'Method and system for generating trusted security labels for electronic documents'. U.S.

Chavan, J. and Desai, P. (2013) 'Relational Data Leakage Detection using Fake Object and Allocation Strategies', *International Journal of Computer Applications*, 80(16), pp. 15–21. doi: 10.1.1.403.2895.

Cheddad, A. *et al.* (2010) 'Digital image steganography: Survey and analysis of current methods', *Signal Processing*, 90(3), pp. 727–752. doi: 10.1016/j.sigpro.2009.08.010.

Clarifai (2018) *Clarifai*. Available at: https://clarifai.com/ (Accessed: 16 March 2019).

Clark, J. W. (2016) 'Threat from Within: Case Studies of Insiders Who Committed Information Technology Sabotage', in *2016 11th International Conference on Availability, Reliability and Security (ARES)*. IEEE, pp. 414–422. doi: 10.1109/ARES.2016.78.

Clarke, N. (2010) *Computer Forensics: A Pocket Guide*. New york: IT Governance Ltd. Available at: http://www.amazon.com/Computer-Forensics-A-Pocket-Guide/dp/1849280398.

Clarke, N. (2011) *Transparent user authentication: biometrics, RFID and behavioural profiling*. Springer Science & Business Media.

Clarke, N. *et al.* (2017) 'Insider Misuse Identification using Transparent Biometrics', *Proceedings of the 50th Hawaii International Conference on System Sciences*, pp. 4031–4040.

Clarke, N., Karatzouni, S. and Furnell, S. (2008) 'Transparent facial recognition

for mobile devices', in *Proceedings of the 7th Security Conference*. Las Vegas. Available at: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Transparent+Facial+Recognition+for+Mobile+Devices#0.

Cohen, M. I., Bilby, D. and Caronni, G. (2011) 'Distributed forensics and incident response in the enterprise', *Digital Investigation*. Elsevier Ltd, 8, pp. S101–S110. doi: 10.1016/j.diin.2011.05.012.

Collins, M. L. *et al.* (2013) *Spotlight On : Insider Theft of Intellectual Property inside the U . S . Involving Foreign Governments or Organizations*, *Intellectual Property*.

Colwill, C. (2009) 'Human factors in information security: The insider threat – Who can you trust these days?', *Information Security Technical Report*. Elsevier Ltd, 14(4), pp. 186–196. doi: 10.1016/j.istr.2010.04.004.

Dalrymple, J. (2013) *Man's child porn charges dismissed to make way for federal case*. Available at: http://www.heraldextra.com/news/local/crime-and-courts/man-s-child-porn-charges-dismissed-to-make-way-for/article_bc5c1cd5-44d1-5e4a-9401-b2e971b1084b.html (Accessed: 1 May 2016).

Damiani, E. *et al.* (2004) 'An Open Digest-based Technique for Spam Detection', *Proceedings of the 2004 International Workshop on Security in Parallel and Distributed Systems*, 1(1), pp. 559–564. Available at: http://spdp.di.unimi.it/papers/pdcs04.pdf.

Davis, J. and Goadrich, M. (2006) 'The relationship between Precision-Recall

and ROC curves', in *Proceedings of the 23rd international conference on Machine learning - ICML '06*. New York, New York, USA: ACM Press, pp. 233–240. doi: 10.1145/1143844.1143874.

Denning, D. E. (1976) 'A lattice model of secure information flow', *Communications of the ACM*, 19(5), pp. 236–243. doi: 10.1145/360051.360056.

Du, D., Yu, L. and Brooks, R. R. (2015) 'Semantic Similarity Detection For Data Leak Prevention', in *Proceedings of the 10th Annual Cyber and Information Security Research Conference on - CISR '15*. New York, New York, USA: ACM Press, pp. 1–6. doi: 10.1145/2746266.2746270.

Dugad, R., Ratakonda, K. and Ahuja, N. (1998) 'A new wavelet-based scheme for watermarking images', in *Proceedings 1998 International Conference on Image Processing. ICIP98 (Cat. No.98CB36269)*. IEEE Comput. Soc, pp. 419–423. doi: 10.1109/ICIP.1998.723406.

Eden, P. *et al.* (2016) 'Forensic Readiness for SCADA/ICS Incident Response', in. doi: 10.14236/ewic/ICS2016.16.

Edward Jero, S., Ramu, P. and Ramakrishnan, S. (2014) 'Discrete Wavelet Transform and Singular Value Decomposition Based ECG Steganography for Secured Patient Information Transmission', *Journal of Medical Systems*, 38(10), p. 132. doi: 10.1007/s10916-014-0132-z.

Elmrabit, N., Yang, S.-H. and Yang, L. (2015) 'Insider threats in information security categories and approaches', in *2015 21st International Conference on Automation and Computing (ICAC)*. IEEE, pp. 1–6. doi:

10.1109/IConAC.2015.7313979.

Fox News (2009) *Framed for Child Porn by a PC Virus*. Available at: http://www.foxnews.com/story/2009/11/09/framed-for-child-porn-by-pc-virus.html (Accessed: 1 May 2009).

Fridrich, J. (2004) 'Feature-Based Steganalysis for JPEG Images and Its Implications for Future Design of Steganographic Schemes', in, pp. 67–81. doi: 10.1007/978-3-540-30114-1_6.

Fridrich, J. (2009) *Steganography in Digital Media*. Cambridge: Cambridge University Press. doi: 10.1017/CBO9781139192903.

Fujikawa, M., Mori, H. and Terada, K. (2014) 'Study of the Detection System for Onscreen Contents Shooting (Countermeasure against Information Leakage by Video Recording/Photo Shooting)', *International Journal of Information and Electronics Engineering*, 4(3). doi: 10.7763/IJIEE.2014.V4.444.

Gartner Inc. (2016) *Gartner Says By 2018 25 Percent of Organizations Will Review Privileged Activity and Reduce Data Leakage Incidents By 33 Percent*. Available at: http://www.gartner.com/newsroom/id/3207217 (Accessed: 18 March 2016).

Gessiou, E., Vu, Q. H. and Ioannidis, S. (2011) 'IRILD: An Information Retrieval Based Method for Information Leak Detection', *2011 Seventh European Conference on Computer Network Defense*, pp. 33–40. doi: 10.1109/EC2ND.2011.21.

Gonzalez-Sosa, E. *et al.* (2018) 'Facial Soft Biometrics for Recognition in the Wild: Recent Works, Annotation, and COTS Evaluation', *IEEE Transactions on Information Forensics and Security*, 13(8), pp. 2001–2014. doi: 10.1109/TIFS.2018.2807791.

GoogleCloudVision (2019) *Google Cloud Vision*. Available at: https://cloud.google.com/vision/ (Accessed: 16 March 2019).

Gov.uk (2015) *Data protection and your business - GOV.UK*. Available at: https://www.gov.uk/data-protection-your-business/monitoring-staff-at-work (Accessed: 20 June 2015).

Grispos, G., Storer, T. and Glisson, W. B. (2012) 'Calm Before the Storm: The Challenges of Cloud Computing in Digital Forensics', *International Journal of Digital Crime and Forensics*, 4(2), pp. 28–48. doi: 10.4018/jdcf.2012040103.

Harris, J. D. *et al.* (2014) 'How to Write a Systematic Review', *The American Journal of Sports Medicine*, 42(11), pp. 2761–2768. doi: 10.1177/0363546513497567.

Hashem, Y. *et al.* (2015) 'Towards Insider Threat Detection Using Psychophysiological Signals', in *Proceedings of the 7th ACM CCS International Workshop on Managing Insider Security Threats - MIST '15*. New York, New York, USA: ACM Press, pp. 71–74. doi: 10.1145/2808783.2808792.

Ho, S. M., Kaarst-Brown, M. and Benbasat, I. (2018) 'Trustworthiness attribution: Inquiry into insider threat detection', *Journal of the Association for Information Science and Technology*, 69(2), pp. 271–280. doi:

10.1002/asi.23938.

Hong, T. (2014) 'A Mantrap-Inspired, User-Centric Data Leakage Prevention (DLP) Approach', *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pp. 1033–1039. doi: 10.1109/CloudCom.2014.23.

Hunker, J. and Probst, C. (2011) 'Insiders and Insider Threats An Overview of Definitions and Mitigation Techniques', *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 2(1), pp. 4–27. Available at: http://isyou.info/jowua/papers/jowua-v2n1-1.pdf.

Huth, C. L. *et al.* (2013) 'Guest editorial: A brief overview of data leakage and insider threats', *Information Systems Frontiers*, 15(1), pp. 1–4. doi: 10.1007/s10796-013-9419-8.

Huth, C. L. (2013) 'The insider threat and employee privacy: An overview of recent case law', *Computer Law & Security Review*. Elsevier Ltd, 29(4), pp. 368–381. doi: 10.1016/j.clsr.2013.05.014.

IS Decisions (2014) *The Insider Threat Security Manifesto: Beating the threat within*. Available at: http://www.isdecisions.com/resources/pdf/insiderthreatmanifesto.pdf (Accessed: 24 April 2016).

IS Decisions (2018) *A STUDY OF INSIDER THREAT PERSONAS*. Available at: https://www.isdecisions.com/insider-threat-persona-study/.

Iyengar, S. S. and Miller, J. (2015) *Biometrics and digital forensics: Cyber*

security connections. Available at: http://www.deccanherald.com/content/515241/biometrics-digital-forensics-cyber-security.html (Accessed: 16 March 2016).

Jackson, J. C., Choi, V. K. and Gelfand, M. J. (2019) 'Revenge: A Multilevel Review and Synthesis', *Annual Review of Psychology*, 70(1), pp. 319–345. doi: 10.1146/annurev-psych-010418-103305.

Jadhav, R. (2012) 'Data leakage detection', *International Journal of Computer Science & Communication Networks*, 03(01), pp. 37–45.

Jain, A. K., Flynn, P. and Ross, A. A. (eds) (2008) *Handbook of Biometrics*. Boston, MA: Springer US. doi: 10.1007/978-0-387-71041-9.

Kale, S. A. and S.V.Kulkarni (2012) 'Data Leakage Detection', *International Journal of Advanced Research in Computer and Communication Engineering*, 1(9), pp. 668–678.

Kalyan, C. and Chandrasekaran, K. (2007) 'Information leak detection in financial e-mails using mail pattern analysis under partial information', in *Proceedings of the 7th WSEAS International Conference on Applied Informatics and Communications*. Athens, Greece, pp. 104–109. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.413.1013.

Katz, G., Elovici, Y. and Shapira, B. (2014) 'CoBAn: A context based model for data leakage prevention', *Information Sciences*. Elsevier Inc., 262(June 2002), pp. 137–158. doi: 10.1016/j.ins.2013.10.005.

Kemerlis, V. P. *et al.* (2010) 'iLeak: A lightweight system for detecting inadvertent information leaks', *Proceedings - European Conference on Computer Network Defense, EC2ND 2010*, pp. 21–28. doi: 10.1109/EC2ND.2010.13.

Khan, M. I., Foley, S. N. and O'Sullivan, B. (2019) 'DBMS Log Analytics for Detecting Insider Threats in Contemporary Organizations', in, pp. 207–234. doi: 10.4018/978-1-5225-5984-9.ch010.

Khan, S. *et al.* (2016) 'Network forensics: Review, taxonomy, and open challenges', *Journal of Network and Computer Applications*. Elsevier, 66, pp. 214–235. doi: 10.1016/j.jnca.2016.03.005.

Kim, Y., Duric, Z. and Richards, D. (2006) 'Modified Matrix Encoding Technique for Minimal Distortion Steganography', in *Information Hiding*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 314–327. doi: 10.1007/978-3-540-74124-4_21.

Kipper, G. (2003) *Investigator's Guide to Steganography*. 1st edn. New York: Auerbach Publications.

Kornblum, J. (2006) 'Identifying almost identical files using context triggered piecewise hashing', *Digital Investigation*, 3(SUPPL.), pp. 91–97. doi: 10.1016/j.diin.2006.06.015.

Koutsourelis, D. and Katsikas, S. K. (2014) 'Designing and developing a free Data Loss Prevention system', in *Proceedings of the 18th Panhellenic Conference on Informatics - PCI '14*. New York, New York, USA: ACM Press, pp. 1–5. doi: 10.1145/2645791.2645833.

Kumar, N. *et al.* (2014) 'Detection of Data Leakage in Cloud Computing Environment', in *2014 International Conference on Computational Intelligence and Communication Networks*. IEEE, pp. 803–807. doi: 10.1109/CICN.2014.172.

Lee, H. *et al.* (2014) 'An Application of Data Leakage Prevention System based on Biometrics Signals Recognition Technology', in *The 3rd International Conference on Networking and Technology.*, pp. 1–5.

Lee, S. *et al.* (2013) 'PDT-BI: Proactive Detection Technology based on the Biometric Information for Preventing Internal Information Leakage', *International Journal of Bio-Science and Bio-Technology*, 5(5), pp. 187–196. doi: 10.14257/ijbsbt.2013.5.5.20.

Lee, Y.-C. *et al.* (2009) 'Information leakage detection in distributed systems using software agents', in *2009 IEEE Symposium on Intelligent Agents*. IEEE, pp. 128–135. doi: 10.1109/IA.2009.4927510.

Liu, J. *et al.* (2018) 'The Reincarnation of Grille Cipher: A Generative Approach'. Available at: http://arxiv.org/abs/1804.06514.

Liu, L. *et al.* (2018) 'Detecting and Preventing Cyber Insider Threats: A Survey', *IEEE Communications Surveys & Tutorials*, 20(2), pp. 1397–1417. doi: 10.1109/COMST.2018.2800740.

Liu, T. *et al.* (2014) 'Towards misdirected email detection for preventing information leakage', in *2014 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, pp. 1–6. doi: 10.1109/ISCC.2014.6912554.

Macq, B., Alface, P. R. and Montanola, M. (2015) 'Applicability of watermarking for intellectual property rights protection in a 3D printing scenario', in *Proceedings of the 20th International Conference on 3D Web Technology - Web3D '15*. New York, New York, USA: ACM Press, pp. 89–95. doi: 10.1145/2775292.2775313.

Magklaras, G., Furnell, S. and Papadaki, M. (2011) 'LUARM – An Audit Engine for Insider Misuse Detection', *International Journal of Digital Crime and Forensics*, 3(3), pp. 37–49. doi: 10.4018/jdcf.2011070103.

Manmadhan, N. *et al.* (2014a) 'Design for Prevention of Intranet Information Leakage via Emails', in *Communications in Computer and Information Science*, pp. 136–148. doi: 10.1007/978-3-662-44966-0_13.

Manmadhan, N. *et al.* (2014b) 'Design for Prevention of Intranet Information Leakage via Emails', in *Security in Computing and Communications*, pp. 136–148. doi: 10.1007/978-3-662-44966-0_13.

Meuwly, D. and Veldhuis, R. (2012) 'Forensic biometrics: From two communities to one discipline', *Biometrics Special Interest Group (BIOSIG), 2012 BIOSIG-Proceedings of the International Conference of the. IEEE, 2012*. Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6313550.

MicrosoftFace (2019) *Microsoft Face API*. Available at: https://azure.microsoft.com/en-gb/services/cognitive-services/face/ (Accessed: 16 March 2019).

Miri, A. and Faez, K. (2018) 'An image steganography method based on integer

wavelet transform', *Multimedia Tools and Applications*, 77(11), pp. 13133–13144. doi: 10.1007/s11042-017-4935-z.

Mordini, E. and Petrini, C. (2007) 'Ethical and social implications of biometric identification technology', *Annali dell'Istituto superiore di sanita*, 47(1), pp. 5–11.

Moshinsky, B. (2017) *LEAKED DOCUMENT: Bank of England has 'significant concern' over post-Brexit approval for Deutsche Bank's UK branch*. Available at: http://uk.businessinsider.com/bank-of-england-document-deutsche-bank-post-brexit-uk-2017-8 (Accessed: 7 September 2017).

MyDLP (2014) *MyDLP*. Available at: https://www.mydlp.com (Accessed: 16 March 2019).

Nelson, M. D. and Xie, M. (2014) 'DATA LEAK PROTECTION'. U.S.

Neurotechnology (2018) *Neurotechnology*. Available at: http://www.neurotechnology.com (Accessed: 16 March 2019).

Nourian, A. and Madnick, S. (2018) 'A Systems Theoretic Approach to the Security Threats in Cyber Physical Systems Applied to Stuxnet', *IEEE Transactions on Dependable and Secure Computing*, 15(1), pp. 2–13. doi: 10.1109/TDSC.2015.2509994.

Oliver, J., Cheng, C. and Chen, Y. (2013) 'TLSH -- A Locality Sensitive Hash', in *2013 Fourth Cybercrime and Trustworthy Computing Workshop*. IEEE, pp. 7–13. doi: 10.1109/CTC.2013.9.

Oliver, J., Forman, S. and Cheng, C. (2014) 'Using Randomization to Attack

Similarity Digests', in, pp. 199–210. doi: 10.1007/978-3-662-45670-5_19.

OpenCV (2019) *OpenCV*. Available at: https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html (Accessed: 16 March 2019).

OpenDLP (2014) *OpenDLP*. Available at: https://code.google.com/p/opendlp/ (Accessed: 16 March 2019).

OpenStego (2017). Available at: https://www.openstego.com/ (Accessed: 16 March 2019).

Palmer, G. (2001) 'A Road Map for Digital Forensic Research (DFRWS)', in *The Digital Forensic Research Conference*.

Papadimitriou, P. and Garcia-Molina, H. (2011) 'Data Leakage Detection', *IEEE Transactions on Knowledge and Data Engineering*, 23(1), pp. 51–63. doi: 10.1109/TKDE.2010.100.

Park, B., Park, J. and Lee, S. (2009) 'Data concealment and detection in Microsoft Office 2007 files', *Digital Investigation*. Elsevier Ltd, 5(3–4), pp. 104–114. doi: 10.1016/j.diin.2008.12.001.

Patzakis, B. J. (2003) 'New Incident Response Best Practices', (September).

Pilli, E. S., Joshi, R. C. and Niyogi, R. (2010) 'Network forensic frameworks: Survey and research challenges', *Digital Investigation*. Elsevier Ltd, 7(1–2), pp. 14–27. doi: 10.1016/j.diin.2010.02.003.

Poljicak, A. (2011) 'Discrete Fourier transform-based watermarking method with an optimal implementation radius', *Journal of Electronic Imaging*, 20(3), p. 033008. doi: 10.1117/1.3609010.

Pressfreedomtracker.us (2019) *Reality Winner charged with leaking information to The Intercept*. Available at: https://pressfreedomtracker.us/all-incidents/reality-winner-charged-leaking-information-intercept/ (Accessed: 20 October 2019).

QL, Z. *et al.* (2018) 'Steganography using reversible texture synthesis based on seeded region growing and LSB', *Comput. Mater. Con*, 55(1), pp. 151–163.

Quick, D. and Choo, K.-K. R. (2013) 'Forensic collection of cloud storage data: Does the act of collection result in changes to the data or its metadata?', *Digital Investigation*. Elsevier Ltd, 10(3), pp. 266–277. doi: 10.1016/j.diin.2013.07.001.

Rafique, M. and Khan, M. N. A. (2013) 'Exploring Static and Live Digital Forensics: Methods, Practices and Tools', *International Journal of Scientific & Engineering Research*, 4(10), pp. 1048–1056.

Rekhis, S. and Boudriga, N. (2012) 'A System for Formal Digital Forensic Investigation Aware of Anti-Forensic Attacks', *IEEE Transactions on Information Forensics and Security*, 7(2), pp. 635–650. doi: 10.1109/TIFS.2011.2176117.

RIPA (2014) *Regulation of Investigatory Powers Act 2000*. Available at: https://www.legislation.gov.uk/ukpga/2000/23/pdfs/ukpga_20000023_en.pdf (Accessed: 16 March 2019).

Roussev, V. (2010) 'Data Fingerprinting with Similarity Digests', in *IFIP Advances in Information and Communication Technology*, pp. 207–226. doi: 10.1007/978-3-642-15506-2_15.

Rutherford, C. (2010) *Man had thousands of child porn images*, *Daily Record*. Available at: htt://www.dailyrecord.co.uk/news/local-news/man-thousands-child-porn-images-2422584 (Accessed: 1 May 2016).

Saevanee, H. *et al.* (2015) 'Continuous user authentication using multi-modal biometrics', *Computers & Security*, 53, pp. 234–246. doi: 10.1016/j.cose.2015.06.001.

Safa, N. S. *et al.* (2018) 'Motivation and opportunity based model to reduce information security insider threats in organisations', *Journal of Information Security and Applications*, 40, pp. 247–257. doi: 10.1016/j.jisa.2017.11.001.

SANS Institute (2016a) *SANS Investigative Forensics Toolkit Documentation*. Release 3.0. Available at: https://media.readthedocs.org/pdf/sift/latest/sift.pdf (Accessed: 12 February 2016).

SANS Institute (2016b) *SANS Investigative Forensics Toolkit Documentation*.

Schaefer, G. and Stich, M. (2003) 'UCID: an uncompressed color image database', in Yeung, M. M., Lienhart, R. W., and Li, C.-S. (eds) *SPIE 5307, Storage and Retrieval Methods and Applications for Multimedia 2004*, pp. 472–480. doi: 10.1117/12.525375.

Shabtai, A., Elovici, Y. and Rokach, L. (2012a) *A Survey of Data Leakage*

Detection and Prevention Solutions. 1st edn, *Springer Science & Business Media*. 1st edn. Boston, MA: Springer US (SpringerBriefs in Computer Science). doi: 10.1007/978-1-4614-2053-8.

Shabtai, A., Elovici, Y. and Rokach, L. (2012b) *A Survey of Data Leakage Detection and Prevention Solutions*. Boston, MA: Springer US (SpringerBriefs in Computer Science). doi: 10.1007/978-1-4614-2053-8.

Shapira, Y., Shapira, B. and Shabtai, A. (2013) 'Content-based data leakage detection using extended fingerprinting', *arXiv preprint arXiv:1302.2028*. Available at: http://arxiv.org/abs/1302.2028.

Shavers, B. (2013) *Placing the suspect behind the keyboard: using digital forensics and investigative techniques to identify cybercrime suspects*. 1st edn. Newnes.

Shetty, J. and Adibi., J. (2004) 'The Enron email dataset database schema and brief statistical report (2004)', *Information sciences institute technical report, University of Southern California*, 4(1), pp. 120–128.

Shields, C., Frieder, O. and Maloof, M. (2011) 'A system for the proactive, continuous, and efficient collection of digital forensic evidence', *Digital Investigation*, 8(SUPPL.), pp. 3–13. doi: 10.1016/j.diin.2011.05.002.

Shih, F. Y. (2017) *Digital Watermarking and Steganography: Fundamentals and Techniques*. CRC press.

Shoichet, C., Botelho, G. and Berlinger, J. (2016) *Brothers ID'd as suicide*

bombers in Belgium, 1 suspect 'on the run', *CNN*. Available at: http://edition.cnn.com/2016/03/23/europe/brussels-investigation/ (Accessed: 5 March 2019).

Sloan, T. and Hernandez-Castro, J. (2018) 'Dismantling OpenPuff PDF steganography', *Digital Investigation*, 25, pp. 90–96. doi: 10.1016/j.diin.2018.03.003.

Smith, R. (2007) 'An Overview of the Tesseract OCR Engine', in *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007) Vol 2*. IEEE, pp. 629–633. doi: 10.1109/ICDAR.2007.4376991.

Soria-Lorente, A. and Berres, S. (2017) 'A Secure Steganographic Algorithm Based on Frequency Domain for the Transmission of Hidden Information', *Security and Communication Networks*, 2017, pp. 1–14. doi: 10.1155/2017/5397082.

Stanton, J. M. *et al.* (2005) 'Analysis of end user security behaviors', *Computers & Security*, 24(2), pp. 124–133. doi: 10.1016/j.cose.2004.07.001.

Stuev, J. (2019) *Former Intelligence Analyst Charged with Disclosing Classified Information*. Available at: https://www.justice.gov/usao-edva/pr/former-intelligence-analyst-charged-disclosing-classified-information (Accessed: 20 October 2019).

The Intercept (no date). Available at: https://theintercept.com/ (Accessed: 5 September 2017).

Titcomb, J. (2017) *WikiLeaks releases thousands of hacked Macron campaign emails*. Available at: http://www.telegraph.co.uk/news/2017/07/31/wikileaks-releases-thousands-hacked-macron-campaign-emails/ (Accessed: 7 September 2017).

Toor, A. S. and Wechsler, H. (2018) 'Biometrics and forensics integration using deep multi-modal semantic alignment and joint embedding', *Pattern Recognition Letters*, 113, pp. 29–37. doi: 10.1016/j.patrec.2017.02.012.

UdhamSingh, K. (2014) 'A Survey on Image Steganography Techniques', *International Journal of Computer Applications*, 97(18), pp. 10–20. doi: 10.5120/17105-7746.

Vaidya, S. (2019) *OpenStego*. Available at: https://www.openstego.com/index.html (Accessed: 5 March 2019).

Valjarevic, A. and Venter, H. (2012) 'Towards Solving the Identity Challenge', in *The 7th International Workshop on Digital Forensics and Incident Analysis (WDFIA 2012)*, pp. 129–138.

Verizon (2018) *2018 Data Breach Investigation Report*. Available at: https://enterprise.verizon.com/resources/reports/dbir (Accessed: 10 May 2019).

Vincze, E. A. (2016) 'Challenges in digital forensics', *Police Practice and Research*, 4263(February), pp. 1–12. doi: 10.1080/15614263.2015.1128163.

Wang, R.-Z., Lin, C.-F. and Lin, J.-C. (2001) 'Image hiding by optimal LSB substitution and genetic algorithm', *Pattern Recognition*, 34(3), pp. 671–683. doi:

10.1016/S0031-3203(00)00015-7.

Wen Gao *et al.* (2008) 'The CAS-PEAL Large-Scale Chinese Face Database and Baseline Evaluations', *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 38(1), pp. 149–161. doi: 10.1109/TSMCA.2007.909557.

Widup, S. (2014) *Computer Forensics and Digital Investigation with EnCase Forensic v7*. 1st edn. McGraw-Hill Osborne.

WikiLeaks.org (2009) *A billion in secret Congressional reports*. Available at: https://wikileaks.org/wiki/Change_you_can_download:_a_billion_in_secret_Congressional_reports (Accessed: 4 September 2017).

WikiLeaks (2017) *WikiLeaks publishes 'biggest ever leak of secret CIA documents'*. Available at: https://www.theguardian.com/media/2017/mar/07/wikileaks-publishes-biggest-ever-leak-of-secret-cia-documents-hacking-surveillance (Accessed: 9 September 2017).

WikiLeaks (no date). Available at: https://wikileaks.org (Accessed: 5 September 2017).

Wu, J.-S. *et al.* (2013) 'Key Stroke Profiling for Data Loss Prevention', in *2013 Conference on Technologies and Applications of Artificial Intelligence*. IEEE, pp. 7–12. doi: 10.1109/TAAI.2013.16.

Xiao, Y. and Watson, M. (2019) 'Guidance on Conducting a Systematic

Literature Review', *Journal of Planning Education and Research*, 39(1), pp. 93-112. doi: 10.1177/0739456X17723971.

Yu, H. and Yang, J. (2001) 'A direct LDA algorithm for high-dimensional data – with application to face recognition', *Pattern Recognition*, 34(10), pp. 2067-2070. doi: 10.1016/S0031-3203(00)00162-X.

Zawoad, S. and Hasan, R. (2013) 'Cloud Forensics: A Meta-Study of Challenges, Approaches, and Open Problems'. Available at: http://arxiv.org/abs/1302.6312.

Appendix A- Experiential Analysis Scripts (Python)

9.5 Null Cipher Embedding and Extracting Script

Mapped to Chapter 6 experimental investigation

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jun 03 19:11:39 2015

@author: aalruban
__author__ = 'aalruban'


"""
How to install the dependencies on a mac
----------------------------------------

1) Reinstall brew.
```bash
rm -rf /usr/local/Cellar /usr/local/.git && brew cleanup
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```
2) Install opencv
```bash
brew tap homebrew/science
brew install opencv
```
3) Add the packages to python
```bash
mkdir -p $HOME/Library/Python/2.7/lib/python/site-packages
echo 'import site; site.addsitedir("/usr/local/lib/python2.7/site-packages")' >>
$HOME/Library/Python/2.7/lib/python/site-packages/homebrew.pth
```
"""

import cv2.cv as cv
import sys

class SteganographyException(Exception):
    pass

class LSBSteg():
    def __init__(self, im):
        self.image = im
        self.width = im.width
        self.height = im.height
        self.size = self.width * self.height
        self.nbchannels = im.channels

        self.maskONEValues = [1,2,4,8,16,32,64,128]
        #Mask used to put one ex:1->00000001, 2->00000010 .. associated with OR bitwise
        self.maskONE = self.maskONEValues.pop(0) #Will be used to do bitwise operations

        self.maskZEROValues = [254,253,251,247,239,223,191,127]
```

```
        #Mak used to put zero ex:254->11111110, 253->11111101 .. associated with AND bitwise
        self.maskZERO = self.maskZEROValues.pop(0)

        self.curwidth = 0 #Current width position
        self.curheight = 0 #Current height position
        self.curchan = 0 #Current channel position

    def saveImage(self,filename):
    # Save the image using the given filename
        cv.SaveImage(filename, self.image)


    def putBinaryValue(self, bits): #Put the bits in the image
        for c in bits:
            val = list(self.image[self.curheight,self.curwidth]) #Get the pixel value as a list
            if int(c) == 1:
                val[self.curchan] = int(val[self.curchan]) | self.maskONE #OR with maskONE
            else:
                val[self.curchan] = int(val[self.curchan]) & self.maskZERO #AND with maskZERO

            self.image[self.curheight,self.curwidth] = tuple(val)
            self.nextSpace() #Move "cursor" to the next space

    def nextSpace(self):#Move to the next slot were information can be taken or put
        if self.curchan == self.nbchannels-1: #Next Space is the following channel
            self.curchan = 0
            if self.curwidth == self.width-1: #Or the first channel of the next pixel of the same line
                self.curwidth = 0
                if self.curheight == self.height-1:#Or the first channel of the first pixel of the next line
                    self.curheight = 0
                    if self.maskONE == 128: #Mask 1000000, so the last mask
                        raise SteganographyException, "Image filled"
                    else: #Or instead of using the first bit start using the second and so on..
                        self.maskONE = self.maskONEValues.pop(0)
                        self.maskZERO = self.maskZEROValues.pop(0)
                else:
                    self.curheight +=1
            else:
                self.curwidth +=1
        else:
            self.curchan +=1

    def readBit(self): #Read a single bit int the image
        val = self.image[self.curheight,self.curwidth][self.curchan]
        val = int(val) & self.maskONE
        self.nextSpace()
        if val > 0:
            return "1"
        else:
            return "0"

    def readByte(self):
        return self.readBits(8)

    def readBits(self, nb): #Read the given number of bits
        bits = ""
        for i in range(nb):
```

```
108             bits += self.readBit()
109         return bits
110
111     def byteValue(self, val):
112         return self.binValue(val, 8)
113
114     def binValue(self, val, bitsize): #Return the binary value of an int as a byte
115         binval = bin(val)[2:]
116         if len(binval) > bitsize:
117             raise SteganographyException, "binary value larger than the expected size"
118         while len(binval) < bitsize:
119             binval = "0"+binval
120         return binval
121
122     def hideText(self, txt):
123         l = len(txt)
124         binl = self.binValue(l, 16) #Length coded on 2 bytes so the text size can be up to 65536 bytes long
125         self.putBinaryValue(binl) #Put text length coded on 4 bytes
126         for char in txt: #And put all the chars
127             c = ord(char)
128             self.putBinaryValue(self.byteValue(c))
129
130     def unhideText(self):
131         ls = self.readBits(16) #Read the text size in bytes
132         l = int(ls,2)
133         i = 0
134         unhideTxt = ""
135         while i < l: #Read all bytes of the text
136             tmp = self.readByte() #So one byte
137             i += 1
138             unhideTxt += chr(int(tmp,2)) #Every chars concatenated to str
139         return unhideTxt
140
141     def hideImage(self, imtohide):
142         w = imtohide.width
143         h = imtohide.height
144         if self.width*self.height*self.nbchannels < w*h*imtohide.channels:
145             raise SteganographyException, "Carrier image not big enough to hold all the datas to
146 steganography"
147         binw = self.binValue(w, 16) #Width coded on to byte so width up to 65536
148         binh = self.binValue(h, 16)
149         self.putBinaryValue(binw) #Put width
150         self.putBinaryValue(binh) #Put height
151         for h in range(imtohide.height): #Iterate the hole image to put every pixel values
152             for w in range(imtohide.width):
153                 for chan in range(imtohide.channels):
154                     val = imtohide[h,w][chan]
155                     self.putBinaryValue(self.byteValue(int(val)))
156
157
158     def unhideImage(self):
159         width = int(self.readBits(16),2) #Read 16bits and convert it in int
160         height = int(self.readBits(16),2)
161         unhideimg = cv.CreateImage((width,height), 8, 3) #Create an image in which we will put all the
162 pixels read
163         for h in range(height):
164             for w in range(width):
```

```
                for chan in range(unhideimg.channels):
                    val = list(unhideimg[h,w])
                    val[chan] = int(self.readByte(),2) #Read the value
                    unhideimg[h,w] = tuple(val)
        return unhideimg

    def hideBin(self, filename):
        f = open(filename,'rb')
        bin = f.read()
        l = len(bin)
        if self.width*self.height*self.nbchannels < l+64:
            raise SteganographyException, "Carrier image not big enough to hold all the datas to
steganography"
        self.putBinaryValue(self.binValue(l, 64))
        for byte in bin:
            self.putBinaryValue(self.byteValue(ord(byte)))

    def unhideBin(self):
        l = int(self.readBits(64),2)
        output = ""
        for i in range(l):
            output += chr(int(self.readByte(),2))
        return output



'''
Methods to expose this functionality to the command-line
'''
def binary_steg_hide(image, binary, result):
    carrier = cv.LoadImage(image)
    steg = LSBSteg(carrier)
    steg.hideBin(binary)
    steg.saveImage(result)

def binary_steg_reveal(steg_image, out):
    inp = cv.LoadImage(steg_image)
    steg = LSBSteg(inp)
    bin = steg.unhideBin()
    f = open(out, "wb")
    f.write(bin)
    f.close()

import argparse

parser = argparse.ArgumentParser(description='This python program applies LSB Steganography to an
image and some type of input')

def main(av):
    bgroup = parser. add_argument_group("Hide binary with steg")
    bgroup.add_argument('-image', help='Provide the original image')
    bgroup.add_argument('-binary', help='The binary file to be obfuscated in the image')
    bgroup.add_argument('-steg-out', help='The resulting steganographic image')

    bgroup = parser.add_argument_group("Reveal binary")
    bgroup.add_argument('-steg-image', help='The steganographic image')
    bgroup.add_argument('-out', help='The original binary')
```

```
222
223     args = parser.parse_args(av[1:])
224
225     if len(av) == 7:
226             binary_steg_hide(args.image, args.binary, args.steg_out)
227     elif len(av) == 5:
228        binary_steg_reveal(args.steg_image, args.out)
229     else:
230        print "Usage: '", av[0], "-h' for help", "\n", args
231
232 if __name__=="__main__":
233     from sys import argv as av
234     main(av)
```

9.6 TLSH Generation and Attacks Against Text Files Script

Mapped to Chapter 8 experimental investigation

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jun 03 19:11:39 2015

@author: aalruban
from __future__ import print_function
import pytesseract
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import sys
import tlsh
import os
import csv
import re, string
import random
from unidecode import unidecode
import shutil
import subprocess
import pickle
import reportlab.pdfbase.ttfonts
from reportlab.lib.units import inch
from reportlab.pdfgen import canvas
from docx import Document
import pandas as pd
import re
import os
import io

from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter
from pdfminer.converter import TextConverter
from pdfminer.layout import LAParams
from pdfminer.pdfpage import PDFPage
from random import randint

from nltk.corpus import wordnet
from nltk.tokenize import word_tokenize
from random import randint
import nltk.data

PUNCTUATION = re.compile('[%s]' % re.escape(string.punctuation))

class Fingerprinter(object):
    '''
    Python implementation of Google Refine fingerprinting algorithm described here:
    https://github.com/OpenRefine/OpenRefine/wiki/Clustering-In-Depth

    Requires the unidecode module: https://github.com/iki/unidecode
    '''
```

```
    def __init__(self, string):
        self.string = self._preprocess(string)

    def _preprocess(self, string):
        '''
        Strip leading and trailing whitespace, lowercase the string, remove all punctuation,
        in that order.
        '''
        return PUNCTUATION.sub('', string.strip().lower())

    def _latinize(self, string):

        return unidecode(string.decode('utf-8'))

    def _unique_preserving_order(self, seq):
        '''
        Returns unique tokens in a list, preserving order. Fastest version found in this
        exercise: http://www.peterbe.com/plog/uniqifiers-benchmark
        '''
        seen = set()
        seen_add = seen.add
        return [x for x in seq if not (x in seen or seen_add(x))]

    def get_fingerprint(self):
        '''
        Gets conventional fingerpint.
        '''
        return self._latinize(' '.join(
            self._unique_preserving_order(
                sorted(self.string.split())
            )
        ))

    def get_ngram_fingerprint(self, n=1):
        '''
        Gets ngram fingerpint based on n-length shingles of the string.
        Default is 1.
        '''
        return self._latinize(''.join(
            self._unique_preserving_order(
                sorted([self.string[i:i + n] for i in range(len(self.string) - n + 1)])
            )
        ))


def getFingerprint(text):
        f = Fingerprinter(text)
        f.get_fingerprint()
        return f.get_ngram_fingerprint(n=1)


def compute_1(path):
    with open(path, 'rb') as f:
        data = f.read()
        hs = tlsh.hash(data)
    return hs

```

```
def compute_2(path):
    h = tlsh.Tlsh()
    with open(path, 'rb') as f:
        for buf in iter(lambda: f.read(512), b''):
            h.update(buf)
    h.final()
    return h


def readDoc(path):
    name = 1

    docList = []

    for filename in os.listdir(path):

        if not (filename.endswith('.txt')):
            continue # skip non-image files and the logo file itself

        doc = path+'/'+filename
            #shutil.move(doc, 'txt/'+str(name).zfill(4)+'.txt')
            docList.append(path+'/'+filename)
            name += 1
    return docList
def docStats():
            path = 'tmp/txt/'
            numOfChar = []
            numOfWords = []
            numOfLines = []
            numOfPars = []
            numOfPages = []

            for filename in os.listdir(path):

                        if not (filename.endswith('.txt')):
                                    continue # skip non-image files and the logo file itself
                        with open('txt/'+filename) as file:
                                    lines = file.read().splitlines()
                                    paras = [value for value in lines[:] if value != '\t']
                                    text = ' '.join(lines)
                                    wordsList = text.split()
                                    chars = ''.join(wordsList)


                                    numOfChar.append(len(chars))
                                    numOfWords.append(len(wordsList))
                                    numOfLines.append(len(lines))
                                    numOfPars.append(len(paras))
                                    numOfPages.append(len(wordsList)/250)

            return [[min(numOfChar), max(numOfChar), sum(numOfChar)/6000], [min(numOfWords),
max(numOfWords), sum(numOfWords)/6000],[min(numOfLines), max(numOfLines),
sum(numOfLines)/6000], [min(numOfPars), max(numOfPars),
sum(numOfPars)/6000],[min(numOfPages), max(numOfPages), sum(numOfPages)/6000]]


def delWordsInc():
```

```
	import os.path
	path = 'tmp/txt/'


	for numWords in range(1,101):
		n = 1
		for filename in range(1,6001):
			print(str(numWords)+':'+str(n))

			filename = str(filename).zfill(4)+'.txt'

			if not (filename.endswith('.txt')):
				continue # skip non-image files and the logo file itself

			with open('tmp/txt/'+filename) as file:
				wordsList = file.read().split()


				for i in range(0, numWords):
					del wordsList[randint(0, len(wordsList)-1)]


				text = ' '.join(wordsList)
			if not os.path.exists('modifiedText/delWordsInc/'+str(numWords).zfill(3)):
				os.makedirs('modifiedText/delWordsInc/'+str(numWords).zfill(3))

			with open('modifiedText/delWordsInc/'+str(numWords).zfill(3)+'/'+filename,
'wb') as f:
				f.write(text)

			if os.path.exists('hash/delWordsInc/'+str(numWords).zfill(3)+'.txt'):
				with open('hash/delWordsInc/'+str(numWords).zfill(3)+'.txt', 'a') as
hashFile:
					hash =
compute_1('modifiedText/delWordsInc/'+str(numWords).zfill(3)+'/'+filename)
					hashFile.write(filename[0:4]+str(hash)+'\n')

			else:
				with open('hash/delWordsInc/'+str(numWords).zfill(3)+'.txt', 'wb') as
hashFile:
					hash =
compute_1('modifiedText/delWordsInc/'+str(numWords).zfill(3)+'/'+filename)
					hashFile.write(filename[0:4]+str(hash)+'\n')



			os.remove('modifiedText/delWordsInc/'+str(numWords).zfill(3)+'/'+filename)

			n += 1


def delWordsInc_test():
	import os.path
	path = 'tmp/txt/'


	for numWords in range(1,2):
```

```
                n = 1
                for filename in range(1,6001):
                        print(str(numWords)+':'+str(n))

                        filename = str(filename).zfill(4)+'.txt'

                        if not (filename.endswith('.txt')):
                                continue # skip non-image files and the logo file itself

                        with open('tmp/txt/'+filename) as file:
                                wordsList = file.read().split()


                                for i in range(0, numWords):
                                        #del wordsList[randint(0, len(wordsList)-1)]
                                        #del wordsList[10]
                                        pass
                                text = ' '.join(wordsList)
                        if not os.path.exists('modifiedText/delWordsInc2/'+str(numWords).zfill(3)):
                                os.makedirs('modifiedText/delWordsInc2/'+str(numWords).zfill(3))

                        with open('modifiedText/delWordsInc2/'+str(numWords).zfill(3)+'/'+filename,
'wb') as f:
                                f.write(text)

                        if os.path.exists('hash/delWordsInc2/'+str(numWords).zfill(3)+'.txt'):
                                with open('hash/delWordsInc2/'+str(numWords).zfill(3)+'.txt', 'a') as
hashFile:
                                        hash =
compute_1('modifiedText/delWordsInc2/'+str(numWords).zfill(3)+'/'+filename)
                                        hashFile.write(filename[0:4]+str(hash)+'\n')

                        else:
                                with open('hash/delWordsInc2/'+str(numWords).zfill(3)+'.txt', 'wb')
as hashFile:
                                        hash =
compute_1('modifiedText/delWordsInc2/'+str(numWords).zfill(3)+'/'+filename)
                                        hashFile.write(filename[0:4]+str(hash)+'\n')




        os.remove('modifiedText/delWordsInc2/'+str(numWords).zfill(3)+'/'+filename)

                        n += 1


def delWordsIncRand():
        path = 'tmp/txt/'
        kList = []
        n = 1

        for filename in os.listdir(path):
                print(n)

                k= randint(1, 101)
                kList.append(k)
```

```
514                     if not (filename.endswith('.txt')):
515                             continue # skip non-image files and the logo file itself
516                     with open('tmp/txt/'+filename) as f:
517                             wordsList = f.read().split()
518
519
520                             for i in range(0, k):
521                                     del wordsList[randint(0, len(wordsList)-1)]
522
523
524                             text = ' '.join(wordsList)
525
526                     with open('modifiedText/delWordsIncRand/'+filename, 'wb') as f2:
527                             f2.write(text)
528
529                     if os.path.exists('hash/delWordsIncRand.txt'):
530                             with open('hash/delWordsIncRand.txt', 'a') as hashFile:
531                                     hash = compute_1('modifiedText/delWordsIncRand/'+filename)
532                                     hashFile.write(filename[0:4]+str(hash)+'\n')
533
534                     else:
535                             with open('hash/delWordsIncRand.txt', 'wb') as hashFile:
536                                     hash = compute_1('modifiedText/delWordsIncRand/'+filename)
537                                     hashFile.write(filename[0:4]+str(hash)+'\n')
538
539
540
541                     os.remove('modifiedText/delWordsIncRand/'+filename)
542
543                     n += 1
544
545             pickle.dump(kList, open('plots/delWordsIncRand.p', "wb"))
546             return kList
547
548     def delWords():
549             path = 'tmp/txt/'
550             kList = []
551             n = 1
552             for filename in os.listdir(path):
553                     print(n)
554                     k= randint(1, 101)
555                     kList.append(k)
556                     if not (filename.endswith('.txt')):
557                             continue # skip non-image files and the logo file itself
558
559
560
561                     with open('tmp/txt/'+filename) as file:
562                             wordsList = file.read().split()
563                             for i in range(0, k):
564                                     del wordsList[randint(0, len(wordsList)-1)]
565
566
567                             text = ' '.join(wordsList)
568                     with open('modifiedText/delWords/'+filename, 'wb') as f:
569                             f.write(text)
570                     n += 1
```

```
571
572             pickle.dump(kList, open('plots/delWords.p', "wb"))
573             return kList
574
575     def swapWords():
576             kList = []
577             n = 1
578             for filename in os.listdir('tmp/txt/'):
579                     print(n)
580                     k= randint(1, 51)
581                     kList.append(k)
582                     if not (filename.endswith('.txt')):
583                             continue # skip non-image files and the logo file itself
584
585                     with open('tmp/txt/'+filename) as file:
586                             wordsList = file.read().split()
587
588                             for itera in range(0, k):
589                                     i = randint(0, len(wordsList)-1)
590                                     j = randint(0, len(wordsList)-1)
591                                     wordsList[i], wordsList[j] = wordsList[j], wordsList[i]
592
593                             #print(len(wordsList))
594                             text = ' '.join(wordsList)
595
596                     if not os.path.exists('modifiedText/swapWords'):
597                             os.makedirs('modifiedText/swapWords')
598
599                     with open('modifiedText/swapWords/'+filename, 'wb') as f2:
600                             f2.write(text)
601
602                     if os.path.exists('hash/swapWords.txt'):
603                             with open('hash/swapWords.txt', 'a') as hashFile:
604                                     hash = compute_1('modifiedText/swapWords/'+filename)
605                                     hashFile.write(filename[0:4]+str(hash)+'\n')
606
607                     else:
608                             with open('hash/swapWords.txt', 'wb') as hashFile:
609                                     hash = compute_1('modifiedText/swapWords/'+filename)
610                                     hashFile.write(filename[0:4]+str(hash)+'\n')
611
612
613                     os.remove('modifiedText/swapWords/'+filename)
614
615
616                     n += 1
617
618             pickle.dump(kList, open('plots/swapWords.p', "wb"))
619             return kList
620
621
622     def wordsSyn():
623
624             # Load a text file if required
625             # Load the pretrained neural net
626             #nltk.download('punkt')
627             #nltk.download('averaged_perceptron_tagger')
```

```
#nltk.download('wordnet')

kList = []
n = 1
tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
for filename in os.listdir('tmp/txt/'):
	output = ""
	print(n)
	k= randint(1, 101)
	kList.append(k)
	if not (filename.endswith('.txt')):
		continue # skip non-image files and the logo file itself

	with open('tmp/txt/'+filename) as file:

		words = file.read().split()


		# Identify the parts of speech

		tagged = nltk.pos_tag(words)
		removedWords = []
		for item in range(0, k):
			ind = randint(0, len(words)-1)
			removedWords.append(ind)

		for i in range(0,len(words)):
		  replacements = []

		  # Only replace nouns with nouns, vowels with vowels etc.
		  for syn in wordnet.synsets(words[i]):

			# Do not attempt to replace proper nouns or determiners
			if tagged[i][1] == 'NNP' or tagged[i][1] == 'DT':
			   break

			if i not in removedWords:
			   break
			# The tokenizer returns strings like NNP, VBP etc
			# but the wordnet synonyms has tags like .n.
			# So we extract the first character from NNP ie n
			# then we check if the dictionary word has a .n. or not


			word_type = tagged[i][1][0].lower()

			if syn.name().find("."+word_type+"."):
			   # extract the word only
			   r = syn.name()[0:syn.name().find(".")]
			   replacements.append(r)

		  if len(replacements) > 0:
			# Choose a random replacement
			replacement = replacements[randint(0,len(replacements)-1)]
			output = output + " " + replacement

		  else:
```

```
                                        # If no replacement could be found, then just use the
                                        # original word
                                        output = output + " " + words[i]

                        if not os.path.exists('modifiedText/wordsSyn'):
                                os.makedirs('modifiedText/wordsSyn')

                        with open('modifiedText/wordsSyn/'+filename, 'wb') as f2:
                                f2.write(output)

                        if os.path.exists('hash/wordsSyn.txt'):
                                with open('hash/wordsSyn.txt', 'a') as hashFile:
                                        hash = compute_1('modifiedText/wordsSyn/'+filename)
                                        hashFile.write(filename[0:4]+str(hash)+'\n')

                        else:
                                with open('hash/wordsSyn.txt', 'wb') as hashFile:
                                        hash = compute_1('modifiedText/wordsSyn/'+filename)
                                        hashFile.write(filename[0:4]+str(hash)+'\n')


                        os.remove('modifiedText/wordsSyn/'+filename)

                        n += 1

        pickle.dump(kList, open('plots/wordsSyn.p', "wb"))
        return kList

def delLines():

        kList = []
        n = 1

        for filename in os.listdir('txt/'):
                print(n)
                k= randint(1, 11)
                kList.append(k)

                if not (filename.endswith('.txt')):
                        continue # skip non-image files and the logo file itself

                with open('txt/'+filename) as file:
                        lines = file.read().splitlines()

                        random_lines = random.sample(lines, k)

                        with open('tmp/cm/'+filename, 'wb') as output_file:
                                        output_file.writelines(line + "\n"
                                                        for line in lines if line not in random_lines)

                        #elif lines: # file is too small
                           #print("\n".join(lines)) # print all lines
                           #with open('modifiedText/delLines/'+filename, 'wb', 0): # empty the file
                                        #pass


                with open('tmp/cm/'+filename) as final_file:
```

```
                wordsList = final_file.read().split()
                text = ' '.join(wordsList)

        with open('modifiedText/delLines/'+filename, 'wb') as modFile: # empty the file
                modFile.write(text)

        n += 1

    pickle.dump(kList, open('plots/delLines.p', "wb"))
    return kList

def linesAllignemt():

    kList = []
    n = 1

    for filename in os.listdir('txt/'):
        print(n)
        k= randint(1, 11)
        if not (filename.endswith('.txt')):
            continue # skip non-image files and the logo file itself

        with open('txt/'+filename) as file:
            lines = file.read().splitlines()

            random_lines = random.sample(lines, k)

            with open('tmp/cm/'+filename, 'wb') as output_file:
                output_file.writelines(line + "\n" for line in lines if line not
in random_lines)

            #elif lines: # file is too small
              #print("\n".join(lines)) # print all lines
              #with open('modifiedText/delLines/'+filename, 'wb', 0): # empty the file
                #pass

        with open('tmp/cm/'+filename) as f1:
            wordsList = f1.read().split()
            text = ' '.join(wordsList)

        if not os.path.exists('modifiedText/delLines'):
            os.makedirs('modifiedText/delLines')

        with open('modifiedText/delLines/'+filename, 'wb') as f2:
            f2.write(text)

        if os.path.exists('hash/delLines.txt'):
            with open('hash/delLines.txt', 'a') as hashFile:
                hash = compute_1('modifiedText/delLines/'+filename)
                hashFile.write(filename[0:4]+str(hash)+'\n')

        else:
            with open('hash/delLines.txt', 'wb') as hashFile:
                hash = compute_1('modifiedText/delLines/'+filename)
                hashFile.write(filename[0:4]+str(hash)+'\n')
```

```
                os.remove('modifiedText/delLines/'+filename)
                n += 1

        pickle.dump(kList, open('plots/delLines.p', "wb"))
        return kList


def swapLines():
        kList = []
        n = 1

        for filename in os.listdir('txt/'):
                print(n)
                k= randint(1, 11)
                kList.append(k)

                if not (filename.endswith('.txt')):
                        continue # skip non-image files and the logo file itself

                with open('txt/'+filename) as file:
                        lines = file.read().splitlines()
                        for itera in range(1, k):
                                i = randint(0, len(lines)-1)
                                j = randint(0, len(lines)-1)
                                lines[i], lines[j] = lines[j], lines[i]
                                text = ' '.join(lines)



                with open('tmp/cm/'+filename, 'wb') as modFile: # empty the file
                        modFile.write(text)

                with open('tmp/cm/'+filename) as final_file:
                        wordsList = final_file.read().split()
                        text = ' '.join(wordsList)

                if not os.path.exists('modifiedText/swapLines'):
                        os.makedirs('modifiedText/swapLines')

                with open('modifiedText/swapLines/'+filename, 'wb') as f2:
                        f2.write(text)

                if os.path.exists('hash/swapLines.txt'):
                        with open('hash/swapLines.txt', 'a') as hashFile:
                                hash = compute_1('modifiedText/swapLines/'+filename)
                                hashFile.write(filename[0:4]+str(hash)+'\n')

                else:
                        with open('hash/swapLines.txt', 'wb') as hashFile:
                                hash = compute_1('modifiedText/swapLines/'+filename)
                                hashFile.write(filename[0:4]+str(hash)+'\n')



                n += 1

        pickle.dump(kList, open('plots/swapLines.p', "wb"))
        return kList
```

```
856
857
858
859 def swapSentences():
860         kList = []
861         n = 1
862         tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
863         for filename in os.listdir('tmp/txt/'):
864                 print(n)
865                 k= randint(1, 6)
866                 kList.append(k)
867                 if not (filename.endswith('.txt')):
868                         continue # skip non-image files and the logo file itself
869
870                 with open('tmp/txt/'+filename) as file:
871                         text = file.read()
872                         sentences = tokenizer.tokenize(text)
873
874                 for itera in range(1, k):
875                                         i = randint(0, len(sentences)-1)
876                                         j = randint(0, len(sentences)-1)
877                                         sentences[i], sentences[j] = sentences[j], sentences[i]
878
879                 text = ' '.join(sentences)
880
881
882                 if not os.path.exists('modifiedText/swapSentences'):
883                         os.makedirs('modifiedText/swapSentences')
884
885                 with open('modifiedText/swapSentences/'+filename, 'wb') as f2:
886                         f2.write(text)
887
888                 if os.path.exists('hash/swapSentences.txt'):
889                         with open('hash/swapSentences.txt', 'a') as hashFile:
890                                 hash = compute_1('modifiedText/swapSentences/'+filename)
891                                 hashFile.write(filename[0:4]+str(hash)+'\n')
892
893                 else:
894                         with open('hash/swapSentences.txt', 'wb') as hashFile:
895                                 hash = compute_1('modifiedText/swapSentences/'+filename)
896                                 hashFile.write(filename[0:4]+str(hash)+'\n')
897
898
899                 os.remove('modifiedText/swapSentences/'+filename)
900
901                 n += 1
902
903         pickle.dump(kList, open('plots/swapSentences.p', "wb"))
904         return kList
905 def delSentences():
906         kList = []
907         n = 1
908         tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
909         for filename in os.listdir('tmp/txt/'):
910                 print(n)
911                 k= randint(1, 11)
912                 kList.append(k)
```

```
913                     if not (filename.endswith('.txt')):
914                             continue # skip non-image files and the logo file itself
915
916                     with open('tmp/txt/'+filename) as file:
917                             text = file.read()
918                             sentences = tokenizer.tokenize(text)
919
920                     for itera in range(1, k):
921                             try:
922                                     del sentences[randint(0, len(sentences)-1)]
923                             except:
924                                     pass
925                     text = ' '.join(sentences)
926
927                     if not os.path.exists('modifiedText/delSentences'):
928                             os.makedirs('modifiedText/delSentences')
929
930                     with open('modifiedText/delSentences/'+filename, 'wb') as f2:
931                             f2.write(text)
932
933                     if os.path.exists('hash/delSentences.txt'):
934                             with open('hash/delSentences.txt', 'a') as hashFile:
935                                     hash = compute_1('modifiedText/delSentences/'+filename)
936                                     hashFile.write(filename[0:4]+str(hash)+'\n')
937
938                     else:
939                             with open('hash/delSentences.txt', 'wb') as hashFile:
940                                     hash = compute_1('modifiedText/delSentences/'+filename)
941                                     hashFile.write(filename[0:4]+str(hash)+'\n')
942
943
944                     os.remove('modifiedText/delSentences/'+filename)
945                     n += 1
946             pickle.dump(kList, open('plots/delSentences.p', "wb"))
947             return kList
948
949     def insertWords():
950
951             kList = []
952             n = 1
953             with open('wordsList.txt') as file:
954                             wordsList = file.read().split()
955
956             tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
957             for filename in os.listdir('tmp/txt/'):
958                     print(n)
959                     k= randint(1, 101)
960                     kList.append(k)
961                     if not (filename.endswith('.txt')):
962                             continue # skip non-image files and the logo file itself
963
964                     with open('tmp/txt/'+filename) as file:
965                             words = file.read().split()
966
967
968                     for itera in range(0, k):
969                             words.insert(randint(0, len(words)-1), wordsList[randint(0, len(wordsList)-1)])
```

```
                text = ' '.join(words)

                if not os.path.exists('modifiedText/insertWords'):
                        os.makedirs('modifiedText/insertWords')

                with open('modifiedText/insertWords/'+filename, 'wb') as f2:
                        f2.write(text)

                if os.path.exists('hash/insertWords.txt'):
                        with open('hash/insertWords.txt', 'a') as hashFile:
                                hash = compute_1('modifiedText/insertWords/'+filename)
                                hashFile.write(filename[0:4]+str(hash)+'\n')

                else:
                        with open('hash/insertWords.txt', 'wb') as hashFile:
                                hash = compute_1('modifiedText/insertWords/'+filename)
                                hashFile.write(filename[0:4]+str(hash)+'\n')


                os.remove('modifiedText/insertWords/'+filename)

                n += 1

        pickle.dump(kList, open('plots/insertWords.p', "wb"))
        return kList

def delLine_range():

        k= 0.20
        for filename in os.listdir('wikileaks-crs-reports'):
        #for filename in os.listdir('test'):
                #print(filename)
                if not (filename.endswith('.txt')):
                        continue # skip non-image files and the logo file itself

                with open('wikileaks-crs-reports/'+filename) as file:
                #with open('test/'+filename) as file:
                        lines = file.read().splitlines()

                removePercentage = int(round(len(lines)*k))
                idx = random.randint(0, len(lines)-removePercentage)

                random_lines = random.sample(lines, int(removePercentage))

                with open('modifiedText/randomRangeLines/20/'+filename, 'w') as output_file:
                        output_file.writelines(line + "\n"
                                for line in lines if line not in lines[idx:idx+removePercentage])
def swapParas():
        kList = []
        n = 1

        for filename in os.listdir('txt/'):
                print(n)
                k= randint(1, 11)
                kList.append(k)
```

```
                if not (filename.endswith('.txt')):
                        continue # skip non-image files and the logo file itself

                with open('txt/'+filename) as file:
                        paragraphs = file.read().splitlines()
                        paragraphs = [value for value in paragraphs[:] if value != '\t']
                        for itera in range(1, k):
                                i = randint(0, len(paragraphs)-1)
                                j = randint(0, len(paragraphs)-1)
                                paragraphs[i], paragraphs[j] = paragraphs[j], paragraphs[i]

                        text = ' '.join(paragraphs)

                with open('tmp/cm/'+filename, 'wb') as modFile:
                        modFile.write(text)

                with open('tmp/cm/'+filename) as final_file:
                        wordsList = final_file.read().split()
                        text = ' '.join(wordsList)

                if not os.path.exists('modifiedText/swapParas'):
                        os.makedirs('modifiedText/swapParas')

                with open('modifiedText/swapParas/'+filename, 'wb') as f2:
                        f2.write(text)

                if os.path.exists('hash/swapParas.txt'):
                        with open('hash/swapParas.txt', 'a') as hashFile:
                                hash = compute_1('modifiedText/swapParas/'+filename)
                                hashFile.write(filename[0:4]+str(hash)+'\n')

                else:
                        with open('hash/swapParas.txt', 'wb') as hashFile:
                                hash = compute_1('modifiedText/swapParas/'+filename)
                                hashFile.write(filename[0:4]+str(hash)+'\n')


                os.remove('modifiedText/swapParas/'+filename)

                n += 1

        pickle.dump(kList, open('plots/swapParas.p', "wb"))
        return kList

def delParas():
        kList = []
        n = 1

        for filename in os.listdir('txt/'):
                print(n)
                k= randint(1, 11)
                kList.append(k)

                if not (filename.endswith('.txt')):
                        continue # skip non-image files and the logo file itself
```

```
                with open('txt/'+filename) as file:
                        paragraphs = file.read().splitlines()
                        paragraphs = [value for value in paragraphs[:] if value != '\t']
                        for itera in range(1, k):
                                del paragraphs[randint(0, len(paragraphs)-1)]

                        text = ' '.join(paragraphs)

                        #print(paragraphs)
                with open('tmp/cm/'+filename, 'wb') as modFile:
                        modFile.write(text)

                with open('tmp/cm/'+filename) as final_file:
                        wordsList = final_file.read().split()
                        text = ' '.join(wordsList)

                if not os.path.exists('modifiedText/delParas'):
                        os.makedirs('modifiedText/delParas')

                with open('modifiedText/delParas/'+filename, 'wb') as f2:
                        f2.write(text)

                if os.path.exists('hash/delParas.txt'):
                        with open('hash/delParas.txt', 'a') as hashFile:
                                hash = compute_1('modifiedText/delParas/'+filename)
                                hashFile.write(filename[0:4]+str(hash)+'\n')

                else:
                        with open('hash/delParas.txt', 'wb') as hashFile:
                                hash = compute_1('modifiedText/delParas/'+filename)
                                hashFile.write(filename[0:4]+str(hash)+'\n')


                os.remove('modifiedText/delParas/'+filename)

                n += 1

        pickle.dump(kList, open('plots/delParas.p', "wb"))
        return kList


def delPara(folder):
        path = 'modifiedText/randomPara/'+folder

        n = 1
        if os.path.exists('tmp'):
                shutil.rmtree('tmp')
        os.mkdir('tmp')
        for filename in os.listdir(path):
                print(n)
                os.mkdir('tmp/'+filename[:-4])
        #for filename in os.listdir('test'):
                #print(filename)
                if not (filename.endswith('.txt')):
                        continue # skip non-image files and the logo file itself

                optimizedPara = []
```

```
                with open(path+'/'+filename, 'r') as file:
                #with open('test/'+filename) as file:
                        paragraphs = file.read().splitlines()
                        paragraphs = (value for value in paragraphs[:] if value != '\t')

                        ITEM =''

                        for para in paragraphs:
                                #print(para)
                                if len(ITEM) < 260 :
                                        ITEM = ITEM+para+' '

                                else:
                                        optimizedPara.append(ITEM)
                                        ITEM = ''

                for index, newPara in enumerate(optimizedPara):
                        with open('tmp/'+filename[:-4]+'/'+str(index).zfill(4)+'.txt', 'w') as f:
                                f.write(newPara)

                hashTocsv(generateHash(readDoc('tmp/'+filename[:-4]+'/')), filename, folder)
                n += 1


def ParaHash2Ways(folder):
        path = 'modifiedText/randomLines/'+folder
        #path = 'wikileaks-crs-reports'

        n = 1
        if os.path.exists('tmp'):
                shutil.rmtree('tmp')
        os.mkdir('tmp')
        for filename in os.listdir(path):
                print(n)
                os.mkdir('tmp/'+filename[:-4])

                if not (filename.endswith('.txt')):
                        continue # skip non-image files and the logo file itself

                optimizedPara = []
                with open(path+'/'+filename, 'r') as file:
                        paragraphs = file.read().splitlines()
                        paragraphs = [value for value in paragraphs[:] if value != '\t']

                        ITEM =''

                        for para in reversed(paragraphs):

                                if len(ITEM) < 260 :
                                        ITEM = ITEM+para+' '

                                else:
                                        optimizedPara.append(ITEM)
                                        ITEM = ''

                for index, newPara in enumerate(optimizedPara):
                        with open('tmp/'+filename[:-4]+'/'+str(index).zfill(4)+'.txt', 'w') as f:
```

```
                                        f.write(newPara)

                    hashTocsv(generateHash(readDoc('tmp/'+filename[:-4]+'/')), folder, filename)
                    n += 1



def delParaFormText():
        path = 'wikileaks-crs-reports'
        k= 0.40

        n = 1
        for filename in os.listdir(path):
                print(n)
                if not (filename.endswith('.txt')):
                        continue # skip non-image files and the logo file itself


                with open(path+'/'+filename, 'r') as file:

                        paragraphs = file.read().splitlines()
                        paragraphs[:] = (value for value in paragraphs[:] if value != '\t')

                removePercentage = int(round(len(paragraphs[:])*k))
                idx = random.randint(0, len(paragraphs[:])-removePercentage)

                random_lines = random.sample(paragraphs[:], int(removePercentage))

                with open('modifiedText/randomPara/40/'+filename, 'w') as output_file:
                        output_file.writelines(line + "\n"
                                for line in paragraphs[:] if line not in random_lines)
                n += 1

def ParaList():
        #path = 'modifiedText/randomPara/'+folder
        path = 'test'

        n = 1
        if os.path.exists('tmp'):
                shutil.rmtree('tmp')
        os.mkdir('tmp')
        for filename in os.listdir(path):
                print(n)
                os.mkdir('tmp/'+filename[:-4])

                if not (filename.endswith('.txt')):
                        continue # skip non-image files and the logo file itself

                optimizedPara = []
                with open(path+'/'+filename, 'r') as file:

                        lines=0
                        words=0
                        characters=0

                        for line in file:
                                wordslist = line.split()
```

```
                                        lines=lines+1
                                        words=words+len(wordslist)
                                        characters += sum(len(word) for word in wordslist)
                                print(lines)
                                print(words)
                                print(characters)
                                paragraphs = file.read().splitlines()
                                paragraphs = (value for value in paragraphs[:] if value != '\t')

                                ITEM =''
                                for para in paragraphs:
                                        if len(ITEM) < 260 :
                                                ITEM = ITEM+para+' '
                                        else:
                                                optimizedPara.append(ITEM)
                                                ITEM = ''
        return optimizedPara


def generateHash(docList):
        n = 1
        hashList= []
        for index, doc in enumerate(docList):
           #letters = ['A', 'B', 'C', 'D', 'E', 'F']

           #delLine(doc)


           hash = compute_1(doc)
           hashList.append(hash)

           #print(len(hash[3:]))



           #for num in numbers:


                #if str(num) not in hash[6:]:
                   #n +=1
                   #print('doc#',n,num)
                   #print(getFingerprint(str(hash[6:])))
                   #for index, letter in enumerate(letters):
                        #if letter in hash[3:]:
                          # print(num,':',letter)
                          # del letters[index]
                          # break
                        #else:
                        #       print(hash[6:])
                        #       print('Not found')
                        #       continue



        return hashList
```

```
def hashTocsv(hashList):


        with open('hash/original/original.txt', 'wb') as f:

                for index, line in enumerate(hashList):

                        f.write(line+'\n')



def wirteTextPDF(docs):

                for filename in os.listdir(docs):
                #for filename in os.listdir('test'):
                        #print(filename)
                        if not (filename.endswith('.txt')):
                                continue # skip non-image files and the logo file itself

                        with open(docs+'/'+filename, 'r') as file:

                                c = canvas.Canvas('pdf      /'+filename[:-4]+'.pdf')
                                c.drawString(100,750, file)
                                c.save()



def text2pdf():
        path = 'txt/'
        k = 1
        for filename in os.listdir(path):
          print(k)
          if not (filename.endswith('.txt')):
                continue # skip non-image files and the logo file itself

          outputfn='pdf/'+filename[:-4]+'.pdf'
          command = str('python txt2pdf.py -o '+outputfn+' '+path+filename)
          print(command)
          print(subprocess.call(command, shell=True))



def tex2docx():
        n = 1
        path = 'cleanedText/'
        for filename in os.listdir(path):
           print(n)
           document = Document()
           #document.add_heading(filename, 0)
           myfile = open(path+filename).read()
           myfile = re.sub(r'[^\x00-\x7F]+|\x0c',' ', myfile) # remove all non-XML-compatible characters
           p = document.add_paragraph(myfile)
           document.save('docx/'+filename[:-4]+'.docx')
           n += 1

def pdf2txt(path):
```

```
    rsrcmgr = PDFResourceManager()
    retstr = io.BytesIO()
    codec = 'utf-8'
    laparams = LAParams()
    device = TextConverter(rsrcmgr, retstr, codec=codec, laparams=laparams)
    fp = open(path, 'rb')
    interpreter = PDFPageInterpreter(rsrcmgr, device)
    password = ""
    maxpages = 0
    caching = True
    pagenos = set()

    for page in PDFPage.get_pages(fp, pagenos, maxpages=maxpages,
                                  password=password,
                                  caching=caching,
                                  check_extractable=True):
        interpreter.process_page(page)

    text = retstr.getvalue()

    fp.close()
    device.close()
    retstr.close()
    myfile = re.sub(r'[^\x00-\x7F]+|\x0c',' ', text) # remove all non-XML-compatible characters
    text2 = "".join([s for s in myfile.splitlines(True) if s.strip("\r\n")])

    text3 = "".join([s.lstrip() for s in text2.splitlines(True) if s.lstrip()])

    return text3

def docx2txt(path):
            document = Document(path)
        newparatextlist = []
        for paratext in document.paragraphs:
            newparatextlist.append(paratext.text.encode("utf-8"))
            text2 = '\n\n'.join(newparatextlist)
            text3 = "".join([s.lstrip() for s in text2.splitlines(True) if s.lstrip()])
        return text3


def text2file(filename):

            if filename[-3:] == 'pdf':
                        #print(pdf2txt(filename))
                        with open('tmp/'+filename[:-4]+'.txt', 'wb') as f:
                                    for line in pdf2txt(filename):
                                                f.write(line.rstrip('\n'))

            elif filename[-4:] == 'docx':
                        with open('tmp/txt/'+filename[:-5]+'.txt', 'wb') as f:
                                    for line in docx2txt(filename):
                                                f.write(line.rstrip('\n'))
            elif filename[-3:] == 'txt':
                        with open('tmp/txt/'+filename[:-4]+'.txt', 'wb') as f:
                                    with open('txt/'+filename, 'rb') as text:
                                                line = "".join([s.lstrip() for s in text.readlines() if s.lstrip()])
                                                line2 = "".join([s.rstrip('\n') for s in line if s.rstrip('\n')])
```

```
                                        f.write(line2)


def tlshHash():
        for doc in range(1,2):
            print(str(doc).zfill(4))
                txt = 'txt/'+str(doc).zfill(4)+'.txt'
                docx = 'docx/'+str(doc).zfill(4)+'.docx'
                pdf = 'pdf/'+str(doc).zfill(4)+'.pdf'

                text2file(txt)
                text2file(pdf)
                #text2file(docx)

                if compute_1('tmp/'+txt[:-3]+'txt') == compute_1('tmp/'+docx[:-4]+'txt') and
compute_1('tmp/'+docx[:-4]+'txt') == compute_1('tmp/'+pdf[:-3]+'txt'):
                        print(1)

                else:
                        print(0)
                print(compute_1('tmp/'+txt[:-3]+'txt'))
                print(compute_1('tmp/'+docx[:-4]+'txt'))

                print(tlsh.diff(compute_1('tmp/'+txt[:-3]+'txt'), compute_1('tmp/'+docx[:-4]+'txt')))

def ocr():

        print(pytesseract.image_to_string(Image.open('test/temp.png')))

#ocr()
#SwapSentence()
#delSentence()
#insertWords()
#swapLines()
#wordsSyn()
#wordsSwap()
#delWordsInc()

def loop():
        n = 1
        for file in os.listdir('txt/'):
                print(n)
                text2file(file)
                n += 1


def plotHist():
        x = swapParas()
        plt.hist(x, bins=20, edgecolor='black',linewidth=1)
        plt.title("Histogram of swapParas")
        plt.xlabel("Number of swapParas")
        plt.ylabel("Frequency")
        plt.savefig('plots/swapParas.png', bbox_inches='tight')
        #plt.show()

def computeTLSH():
```

```
1483            #with open('hash/swapParas.txt', 'wb') as f:
1484            with open('hash/original.txt', 'wb') as f:
1485                    for file in range(1,6001):
1486                            #print(str(k)+':'+str(file))
1487                            filename = str(file).zfill(4)+'.txt'
1488                            #text2file('txt/'+filename)
1489                            hash = compute_1('tmp/txt/'+filename)
1490                            f.write(filename[0:4]+str(hash)+'\n')
1491
1492    #computeTLSH()
1493    #delWordsIncRand()
1494    #delWordsInc()
1495    #swapWords()
1496    #wordsSyn()
1497    #linesAllignemt()
1498    #swapLines()
1499    #delSentences()
1500    #swapParas()
1501    #delParas()
1502    def joinWords():
1503            size = []
1504            for file in range(1,6001):
1505                    filename = str(file).zfill(4)+'.txt'
1506                    #text2file('txt/'+filename)
1507                    size.append(os.stat('txt/'+filename).st_size/1000)
1508            #print(sorted(size))
1509            #count, division = np.histogram(size, bins=3, range=[1, 300])
1510            #for index, c in enumerate(count):
1511                    #print(c, division[index])
1512
1513            plt.hist(size, bins=3, edgecolor='w', range=[1,300])
1514            plt.show()
1515
1516    def computeAvgDiff(output):
1517
1518            diffList = []
1519            filename = str(file).zfill(2)+'.txt'
1520            originalTLSH = pd.read_csv('hash/original.txt', header=None)
1521            ModifiedTLSH = pd.read_csv('hash/delWordsIncRand.txt', header=None)
1522
1523            for index, row in originalTLSH.iterrows():
1524                    #print(index+1)
1525                    hex1 = row[0][4:]
1526                    hex2 = [0][4:]
1527
1528                    diffList.append(tlsh.diff(hex1, hex2))
1529
1530            data = str(file)+','+str(min(diffList))+','+str(max(diffList))+','+str(sum(diffList)/6000)+'\n'
1531            output.write(data)
1532
1533
1534    #for k in range(42,90):
1535            #delWordsInc(k)
1536            #computeTLSH(str(k).zfill(2))
1537    #stat = docStats()
1538    #for list in stat:
1539    #       print(list)
```

```
1540
1541    #plotHist()
1542    #computeTLSH()
1543    with open('result/retrivel/HAShes_diff.csv', 'w') as output:
1544            computeAvgDiff(output)
1545
1546    #print(len(compute_1('wordsList.txt')))
1547    #print('The total number of documents is:', len(docList))
1548    #path = 'modifiedText/randomLines/5'
1549    #ParaHash2Ways()
1550    #hashTocsv(generateHash(readDoc(path)))
1551    #hashTocsv(generateHash(readDoc(path)))
1552    #hex1 = compute_1(sys.argv[1])
1553    #print(len(hex1))
1554    #print('tlsh.hash hex1', hex1)
1555    #hex2 = compute_1(sys.argv[2])
1556    #print('tlshcd .hash hex2', hex2)
1557    #print('tlsh.diff(hex1, hex2)', tlsh.diff(hex1, hex2))
1558    #print('tlsh.diff(hex2, hex1)', tlsh.diff(hex2, hex1))
1559    #h1 = compute_2(sys.argv[1])
1560    #hex1 = h1.hexdigest()
1561    #print('tlsh.Tlsh hex1', hex1)
1562    #h2 = compute_2(sys.argv[2])
1563    #hex2 = h2.hexdigest()
1564    #print('tlsh.Tlsh hex2', hex2)
1565    #print('h1.diff(h2)', h1.diff(h2))
1566    #print('h2.diff(h1)', h2.diff(h1))
1567    #print('h1.diff(hex2)', h1.diff(hex2))
1568    #print('h2.diff(hex1)', h2.diff(hex1))
1569    #h3 = tlsh.Tlsh()
1570    #h3.fromTlshStr(hex2)
1571    #print('tlsh.Tlsh.fromTlshStr', hex2)
1572    #print('h3.diff(h2)', h3.diff(h2))
1573
```

9.7 Grille Cipher Mapping and Retrieving Script

Mapped to chapter 7 and 8 experimental investigations

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jun 03 19:11:39 2015

@author: aalruban
"""
import hashlib
from timeit import default_timer as timer
import collections, binascii #
import numpy as np
import time
from tqdm import tqdm #for counter the timer and status bar
from os.path import basename
import itertools
import datetime
import sqlite3
import sys, os
import cPickle as pickle
import openxmllib
from PyPDF2 import PdfFileReader
import pandas as pd
import shutil

def db(insertDB):
    conn = sqlite3.connect('Bio.db')

    c = conn.cursor()
    #check weather the table is exist or not
    tb_exists = "SELECT name FROM sqlite_master WHERE type='table' AND name='mappedBio'"
    if not conn.execute(tb_exists).fetchone():
        c.execute('''CREATE TABLE mappedBio
                (objname, mappingddate, objhash, author, creator, creationdate, moddate, biobytes,
mappedbytesindex)''')

    # Insert a row of data
    c.execute("INSERT INTO mappedBio VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?);", (insertDB))

    # Save (commit) the changes
    conn.commit()

    # We can also close the connection if we are done with it.
    # Just be sure any changes have been committed or they will be lost.
    conn.close()


# computing hash value of the object, in order to use it for comparison later when we want to make sure
that the
# object has not been modified after we have created the record
def gethash(user_object):
    #print "Calculating", user_object, "hash..."
```

```
    hashx = [hashlib.md5(open(user_object, 'rb').read()).hexdigest()]
    return hashx


# this function return object's bytes as a list
def byte_target(user_object):
    #print "Loading", user_object, "bytes..."
    with open(user_object, "rb") as f:
        # defining the list that we want to store the object's bytes in
        tlist = []
        while True:
            # [1] means to read the object byte by byte
            b = f.read(1)
            # get file bytes
            t = "%s" % (binascii.hexlify(b))
            # when reached the end of the object it breaks
            if not b:
                break
            # add this byte to the list
            tlist.append(t)

    #print tlist
    return tlist

def obj_char(user_object):
    with open(user_object) as f:
            line = f.read().split()

            char_list = [c for c in line[0]]

    return char_list


# this function return Bio source file bytes as a list
def bio(bioSource):
    with open(bioSource) as f:
        #print "Loading", bioSource, "bytes..."
        # defining the list that we want to store the object's bytes in
        biolist = []
        while True:
            # [1] means to read the object byte by byte
            b = f.read(1)
            # get file bytes
            t = "%s" % (binascii.hexlify(b))
            # when reached the end of the object it breaks
            if not b:
                break
            # add this byte to the list
            biolist.append(t)
            # print b
    #print biolist
    return biolist


def bio_char(bioSource):
```

```
    with open(bioSource) as f:
        line = f.read().split()

        char_list = [c for c in line[0]]

    return char_list


# def bio(bioSource):
#     with open(bioSource, "rb") as f:
#         #print "Loading", bioSource, "bytes..."
#         # defining the list that we want to store the object's bytes in
#         biolist = []
#         while True:
#             # [1] means to read the object byte by byte
#             b = f.read(1)
#             # get file bytes
#             t = ' '.join(format(ord(x), 'b') for x in b)
#             # when reached the end of the object it breaks
#             if not b:
#                 break
#             # add this byte to the list
#             biolist.append(t)
#             # print b
#     return biolist


# this function compares objects bytes with  Bio source file bytes and returns matched objects bytes'
index as list
def mapping_1st_match(user_object, bioSource):
    #print "Mapping", bioSource, "with", user_object, "..."
    #print "Mapping..."
    # defining the list that we want to store the matched objects bytes in
    m = []
    # for each byte in the bio byte list
    for z in bio_byte_final:
        # for each byte in the object byte list
        for index, i in enumerate(byte_target_final):
            #if bytes are equal
            if i == z:
                # add the index to the list
                tostr = byte_target_final.index(i)

                m.append(str(tostr))

                break
    # to make sure that all bio bytes are mapped
    # if the length of the mapped list = the length of the bio source bytes list, that means all the bio bytes
has matched bytes in the object
    if len(m) == len(bio_byte_final):
        print("bytes mapping succeeded :)")
    # else, there is a shortage in the object bytes which they are 'lendiff'
    else:
        lendiff = len(bio_byte_final) - len(m)
        print("bytes mapping NOT succeeded,", lendiff, "bytes is needed!")

    return m
```

```
1739
1740
1741    # this function compares objects bytes with Bio source file bytes and returns matched objects bytes' index
1742    as list
1743    def nLocations(object_, bioSource):
1744        #print "Mapping", bioSource, "with", user_object, "..."
1745        #print "Mapping..."
1746        # defining the list that we want to store the matched objects bytes in
1747
1748        # for each byte in the bio byte list
1749
1750        c = 0
1751        m = []
1752
1753        for z in tqdm(bioSource, desc='mapping', leave=True):
1754            #
1755            # if z == '101100':
1756            #     m.append([","])
1757            # elif z == '101110':
1758            #     m.append(["."])
1759
1760            # for each byte in the object byte list
1761            indexes = [i for i, x in enumerate(object_) if x == z]
1762            m.append(indexes)
1763
1764
1765
1766        # to make sure that all bio bytes are mapped
1767        # if the length of the mapped list = the length of the bio source bytes list, that means all the bio bytes
1768    has matched bytes in the object
1769        if len(m) == len(bioSource):
1770
1771            print("bytes mapping succeeded :)")
1772        # else, there is a shortage in the object bytes which they are 'lendiff'
1773        else:
1774            lendiff = len(bioSource) - len(m)
1775            print("bytes mapping NOT succeeded,", lendiff, "bytes is needed!")
1776
1777        return m
1778
1779
1780    # this function return metadata info when the object is a PDF document
1781    def pdfMetadata(user_object):
1782        # load pdf
1783        pdf = PdfFileReader(open(user_object, 'rb'))
1784        # retrieve pdf metadata
1785        docinfo = pdf.getDocumentInfo()
1786        # get author
1787        docAuthor = docinfo.author
1788        # if there is  no author, make it 'None'
1789        if docAuthor == "":
1790            docAuthor = 'None'
1791        else:
1792            docAuthor = docinfo.author
1793        # get creator name
1794        docCreator = docinfo.creator
1795        # get creation date
```

```
    docCreationDate = docinfo['/CreationDate']
    # get modified date
    docModDate = docinfo['/ModDate']
    # creat a list
    MetaData = [docAuthor, docCreator, docCreationDate, docModDate]
    return MetaData


# this function return metadata info when the object is an office document include, docx, xlsx, and
pptx(not tested yet)
def docxMetadata(user_object):
    # load document
    doc = openxmllib.openXmlDocument(path=user_object)
    # load function
    doc2 = doc.coreProperties
    # get creator
    docCreator = doc2['creator']
    # get the name of the person who is the last modified it
    doclastmodBy = doc2['lastModifiedBy']
    # get creation date
    docCreationDate = doc2['created']
    # get modified date
    docModDate = doc2['modified']
    # create a list
    MetaData = [docCreator, doclastmodBy, docCreationDate, docModDate]
    return MetaData


def finalrecord():
    if user_object[-3:] == "pdf":
        # the record is stored as a list right now (in a database later on), the list item order as follow:
        # object title, current system time, object hash value, metadata(docAuthor, docCreator,
docCreationDate, docModDate), bio bytes(spaceless), matched index
        record = filenameAsText + ttime + gethash(user_object) + pdfMetadata(user_object) + Biobits +
convertedtodb
        # to print the first 7 items in the list
        #print "Object Record:", record[:7]
    elif user_object[-4:] == "docx" or user_object[-4:] == "xlsx" or user_object[-4:] == "pptx":
        # object title, current system time, object hash value, metadata(docCreator, doclastmodBy,
docCreationDate, docModDate) , bio bytes(spaceless), matched index
        record = filenameAsText + ttime + gethash(user_object) + docxMetadata(user_object) + Biobits +
convertedtodb
        # to print the first 7 items in the list
        #print "Object Recored:", record[:7]
    else:
        # object title, current system time, object hash value, bio bytes(spaceless), matched index
        record = filenameAsText + ttime + gethash(user_object) + nonelist + Biobits + convertedtodb
        # to print the first 4 items in the list
        #print "Object Record:", record[:7]
    return record


def extract_bio(byte_target_final, mappedBio):
    #print "Extracting", bioSource, "from", user_object, "..."
    # defining the list that we want to store the matched objects bytes in
    m = []
```

```
    #m =  [i for index, i in enumerate(byte_target_final) for j in mappedBio if index == j]

    #for each byte index in the bio
    for z in mappedBio:

        # for each byte index in the object
        for index, i in enumerate(byte_target_final):

            # if indexes are equal
            if index == z:
                # add the byte to the list
                m.append(i)
                break
    return m


def validation(featureVector, byte_target_final, bioSource):
    #this function checks how many imprints can be retiereved
    counter = [] #conut number of success retireved
    lengtbar = len(featureVector)


    for index, i in tqdm(enumerate(featureVector), desc='Number of imprints',total=lengtbar, leave=True):

        if str(extract_bio(byte_target_final, i)) == bioSource:
        # when the retived values are equall add 1 to the counter

            counter.append(index)


    print("")
    print("%s out of %s retrieved" % (len(counter), len(featureVector)))
    #the function returns the number of success retreived
    return [len(counter),len(featureVector)]
    # return

def validationFrequncy(featureVector, byte_target_final, bioSource):
    m = []

    #m =  [i for index, i in enumerate(byte_target_final) for j in mappedBio if index == j]

    #for each byte index in the bio









        # for each byte index in the object
    #     for index, i in enumerate(byte_target_final):
    #
    #         # if indexes are equal
    #         if index == z:
    #             # add the byte to the list
    #             m.append(i)
    #             break
    return m
```

```
def Vm(attack, bio_byte_final):
        n = 1
        retrivedImprints = []
        allelipsidtime = []
        totalimprints = []
        #path = "images/modified/dataset"

        path = "TLSH_FILES/"+attack+'/'
        for filename in os.listdir(path):
            if not (filename.endswith('.txt')):
                continue

            filenameWithoutJpg = os.path.splitext(filename)[0]
            path2 = "imprints/original/"+filenameWithoutJpg+".p"

            byte_target_final = obj_char('TLSH_FILES/'+attack+'/'+str(filename))
            featureVector = pickle.load(open(path2, "rb"))

            print(n)
            start = timer()
            valditionreturn = validation(featureVector, byte_target_final, str(bio_byte_final))
            retrivedImprints.append(valditionreturn)
            end = timer()
            elipsidtime = end - start
            allelipsidtime.append(elipsidtime)
            totalimprints.append(valditionreturn[1])

            n += 1

        print(sum(allelipsidtime))
        print(sum(totalimprints))
        print((sum(allelipsidtime)/sum(totalimprints)))

        pickle.dump(retrivedImprints, open("result/retrivel/"+attack+".p", "wb"))




        shutil.rmtree('TLSH_FILES/'+attack)

def result(attack):
        zero_mapped, total_retrived, total_mapped,lost, total_document_retrived,
total_document_retrived_perentage = [], [], [], [], [], []
        for file in range(1, 2):

                featureVector = pickle.load(open('result/retrivel/'+attack+'.p', "rb"))

                n = 0

                m, x, y = [], [], []
                #print(featureVector)
                for i in featureVector:
                    #print(i[0])
                    #print(i[1])
                    try:
                            i01 = float(i[0]) / float(i[1])
```

```
                        m.append(i01*100)
                    except ZeroDivisionError:
                        n += 1
                    x.append(i[0])
                    y.append(i[1])


                null = []
                for i in m:
                    if i == 0.0:
                        null.append(i)

                zero_mapped.append(n)
                total_retrived.append(sum(x))
                total_mapped.append(sum(y))
                lost.append(len(null))
                total_document_retrived.append(len(featureVector)-len(null))


        result_dict = ({'zero_mapped':zero_mapped, 'total_retrived':total_retrived,
'total_mapped':total_mapped, 'lost':lost, 'total_document_retrived':total_document_retrived})

        result_df = pd.DataFrame.from_dict(result_dict)
        result_df.index = np.arange(1,len(result_df)+1)
        result_df['total_document_retrived_perentage'] = ((6000-result_df['lost'])/6000)*100

        result_df.to_csv('result/retrivel/'+attack+'.csv')


def readText(path, folder):
    text = pd.read_csv('hash/delWordsInc/'+folder+'.txt', header=None)
    for index, row in text.iterrows():
        if not os.path.exists('TLSH_FILES/delWordsInc/'+folder+'/'):
            os.makedirs('TLSH_FILES/delWordsInc/'+folder+'/')

        with open('TLSH_FILES/delWordsInc/'+folder+'/'+str(int(row[0][0:4]) - 1).zfill(4) + '.txt', 'w') as
f:
            f.write(row[0][6:])

def readText_sigleHashFile(attack):
    text = pd.read_csv('hash/'+attack+'.txt', header=None)
    for index, row in text.iterrows():
        if not os.path.exists('TLSH_FILES/'+attack+'/'):
            os.makedirs('TLSH_FILES/'+attack+'/')

        with open('TLSH_FILES/'+attack+'/'+str(int(row[0][0:4]) - 1).zfill(4) + '.txt', 'w') as f:
            f.write(row[0][6:])



#     for k in range(35, 90):
#         print k
#         if not os.path.exists('TLSH_FILES/delWordsInc_70/' + str(k).zfill(2)):
#             os.makedirs('TLSH_FILES/delWordsInc_70/' + str(k).zfill(2))
#         text = pd.read_csv('result/hash/delWordsInc_70/'+str(k).zfill(2)+'.txt', header=None)
#         for index, row in text.iterrows():
```

```
#               dest = 'TLSH_FILES/delWordsInc_70/' + str(k).zfill(2) + '/' + str(int(row[0][0:4]) - 1).zfill(4)
+ '.txt'
#               with open(dest, 'w') as f:
#                   f.write(row[0][6:])

if __name__ == '__main__':
    # define a time object in order to calculate process time
    t0 = time.clock()
    now = datetime.datetime.now()

    if len(sys.argv) < 1 or len(sys.argv) > 4 or sys.argv == "-help" or sys.argv == "-h":
        print("")
        print("Usage: python binariesMapping.py <mapping method> <object path> <Biometric feature
vector file>")
        print("e.g: Usage: python binariesMapping.py --nL objects/obj.jpg objects/fv.txt")
        print("Supported mapping methods:")
        print("  -fT    :first matched location only")
        print("  -nL    :mappes every feature to n-locations for one object")
        print("  -nLm   :mappes every feature to n-locations for list of objects")
        print("  -V     :retireves mapped imprints from a given object")
        print("  -Vm    :retireves mapped imprints from list of object")
        print("  -nT    :mappes feature vector n-times throughout object")

        # define the target file which is the user object that we want to hide Bio inside
        sys.exit()

    user_object = sys.argv[2]

    # define Bio source file, for this experiment I just assumed that we got the model file from function
    bioSource = sys.argv[3]

    # load functions into variables are 10x faster than using functions directly
    # define variables
    # object bytes list
    byte_target_final = byte_target(user_object)
    # bio bytes list
    #bio_byte_final = bio(bioSource)

    bio_byte_final = bio_char(bioSource)
    # get object name to be used in the record
    filenameAsText = [user_object]

    # replacing bio bytes list into spaceless to be used the record for comparison later on
    Biobits = [''.join(bio_byte_final)]

    # formatting time
    currenttime = now.strftime("%Y-%m-%d %H:%M:%S")
    # defining time object to be stored in the record
    ttime = [currenttime]
    nonelist = ['None', 'None', 'None', 'None']
    # because object extension may vary, I use this condition to make sure we get the right meta data

    if sys.argv[1] == ("-fT"):
        cvrtmappied = mapping_1st_match(byte_target_final, bio_byte_final)
        #in order to isert the mapped indexes into the db, I converted ',' into smt else such as '-' to be inserted
into 'mappedbytesindex' colunm
        print(cvrtmappied)
```

```
        convertedtodb = ['-'.join(cvrtmappied)]
        # to remove '-' sign
        print(convertedtodb)


    elif sys.argv[1] == '-nL':
        # this argument will imprints a given objects
        cvrtmappied = nLocations(byte_target_final, bio_byte_final)

        print(cvrtmappied)

        featureVector = map(list, zip(*cvrtmappied))
        #this generates the imprints from each character list


        print("")
        #print featureVector
        print(featureVector)
        print(len(featureVector), "imprints generated")

        pickle.dump(featureVector, open("imprints.p", "wb"))
        # to save the generated imprints into a file



    elif sys.argv[1] == '-nLm':
        # this argument will imprints list of given objects
        #variable = "dataset"
        # this where the objects are located
        path = "TLSH_FILES/original/"
        n = 0
        imprints = []
        allimprint = []
        allelipsidtime = []
        totalimprints = []

        for filename in os.listdir(path):
            #for each object on the givin directory
            if not (filename.endswith('.txt')):
                #ignor any object other than specified
                continue

            #byte_target_final = byte_target(path+filename)
            #  this variable stores object's Hex as a list
            byte_target_final = obj_char(path+filename)


            start = timer()

            cvrtmappied = nLocations(byte_target_final, bio_byte_final)
            #  this variable stores mapped indexes
                #print(cvrtmappied)

            end = timer()

            elipsidtime = end - start
```

```
            allelipsidtime.append(elipsidtime)
            featureVector = map(list, zip(*cvrtmapped))
                #print(featureVector)
            #this generates the imprints from each character list
            totalimprints.append(len(featureVector))
            print(n,"object:", filename, len(featureVector), "imprints generated")
            filenameWithoutJpg = os.path.splitext(filename)[0]
            path2 = "imprints/original/"+filenameWithoutJpg+".p"

            pickle.dump(featureVector, open(path2, "wb"))


            n+=1
            imprints.append(len(featureVector))
            allimprint.append(featureVector)
        print(sum(allelipsidtime))
        print(sum(totalimprints))
        print((sum(allelipsidtime)/sum(totalimprints)))
        pickle.dump(imprints, open("result/original.p", "wb"))
        # to write each obect imprints into a file
        pickle.dump(allimprint, open("result/original_All.p", "wb"))
        # to write all the imprints into a single file

    elif sys.argv[1] == '-V':
        featureVector = pickle.load(open("imprints.p", "rb"))
        validation(featureVector, byte_target_final, str(bio_byte_final))

    elif sys.argv[1] == '-Vm':

        n = 1
        retrivedImprints = []
        allelipsidtime = []
        totalimprints = []
        #path = "images/modified/dataset"
        folder = 'delWordsInc'
        path = "TLSH_FILES/"+folder+"/"
        for filename in os.listdir(path):
            if not (filename.endswith('.txt')):
                continue

            filenameWithoutJpg = os.path.splitext(filename)[0]
            path2 = "imprints/delWordsInc/"+filenameWithoutJpg+".p"

            byte_target_final = obj_char("TLSH_FILES/"+folder+"/"+str(filename))
            featureVector = pickle.load(open(path2, "rb"))

            print(n)
            start = timer()
            valditionreturn = validation(featureVector, byte_target_final, str(bio_byte_final))
            retrivedImprints.append(valditionreturn)
            end = timer()
            elipsidtime = end - start
            allelipsidtime.append(elipsidtime)
            totalimprints.append(valditionreturn[1])

            n += 1
```

```
        print(sum(allelipsidtime))
        print(sum(totalimprints))
        print((sum(allelipsidtime)/sum(totalimprints)))

        pickle.dump(retrivedImprints, open("result/retrivel/"+folder+".p", "wb"))
        # for k in range(34, 90):
        #     n = 1
        #     retrivedImprints = []
        #     allelipsidtime = []
        #     totalimprints = []
        #     # path = "images/modified/dataset"
        #     folder = 'delWordsInc_70/'+str(k).zfill(2)
        #     path = "TLSH_FILES/" + folder + "/"
        #     for filename in os.listdir(path):
        #         if not (filename.endswith('.txt')):
        #             continue
        #
        #         filenameWithoutJpg = os.path.splitext(filename)[0]
        #         path2 = "imprints/original/" + filenameWithoutJpg + ".p"
        #
        #         byte_target_final = byte_target("TLSH_FILES/" + folder + "/" + str(filename))
        #         featureVector = pickle.load(open(path2, "rb"))
        #
        #         print n
        #         start = timer()
        #         valditionreturn = validation(featureVector, byte_target_final, str(bio_byte_final))
        #         retrivedImprints.append(valditionreturn)
        #         end = timer()
        #         elipsidtime = end - start
        #         allelipsidtime.append(elipsidtime)
        #         totalimprints.append(valditionreturn[1])
        #
        #         n += 1
        #
        #     print sum(allelipsidtime)
        #     print sum(totalimprints)
        #     print (sum(allelipsidtime) / sum(totalimprints))
        #
        #     pickle.dump(retrivedImprints, open("result/result/" + folder + ".p", "wb"))

    elif sys.argv[1] == '-nF':
        cvrtmapped = nLocations(byte_target_final, bio_byte_final)
        pickle.dump(cvrtmapped, open("imprints.p", "wb"))


    elif sys.argv[1] == '-Vf':
        cvrtmapped = pickle.load(open("imprints.p", "rb"))


        #print cvrtmapped
        m = []
        for i in cvrtmapped:
            counter = collections.Counter(extract_bio(byte_target_final, i))

            print(counter)

            print("")
```

```
        print("")

elif sys.argv[1] == '-p':
          #for folder in range(1,101):

         #readText(sys.argv[2], str(folder).zfill(3))
              #Vm(str(folder).zfill(3), bio_byte_final)
           attack = 'delSentences'
           readText_sigleHashFile(attack)
           Vm(attack, bio_byte_final)
           result(attack)

    #featureVector = map(list, zip(*cvrtmappied))

    #print len(featureVector), "imprints generated"
    #pickle.dump(featureVector, open("imprints.p", "wb"))

    # insertDB = finalrecord()
    # print "inserting data into database"
    # db(insertDB)
    #
    # assert isinstance(t0, object)
    # print time.clock()
```

End of Thesis